Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Ilenič

Globoki modeli avtorstva umetniških slik

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Blaž Zupan

Ljubljana, 2017

University of Ljubljana
Faculty of Computer and Information Science

Nejc Ilenič

Deep Models of Painting Authorship

master's thesis

the 2nd cycle master's study programme
computer and information science

Advisor: Prof. Dr. Blaž Zupan

Ljubljana, 2017

Declaration of Masters Thesis authorship

I, the undersigned Nejc Ilenič am the author of the Master Thesis entitled:

Deep Models of Painting Authorship

With my signature, I declare that:

- the submitted Thesis is my own unaided work under the supervision of Prof. Dr. Blaž Zupan,
- all electronic forms of the Masters Thesis, title (Slovenian, English), abstract (Slovenian, English) and keywords (Slovenian, English) are identical to the printed form of the Masters Thesis,
- I agree with the publication of the electronic form of the Masters Thesis in the collection "Dela FRI".

In Ljubljana, 15. September 2017 Author's signature:

Acknowledgments

I would like to thank Prof. Dr. Blaž Zupan for making me enthusiastic about machine learning and for enabling me to participate in predictive modelling competitions. Furthermore, I would like to express my gratitude to other members of the Bioinformatics Laboratory for their help and guidance during the thesis. Special thanks go to my friends and family, who were very supportive throughout the course of my studies.

Nejc Ilenič, 2017

Contents

Povzetek

Vse več raziskav se osredotoča na problem avtomatskega prepoznavanja slikarjev iz digitaliziranih umetniških slik. V tem delu se omenjenega izziva lotimo na nadzorovan način, z uporabo konvolucijske nevronske mreže, ki je zaradi visoke izrazne moči sposobna napovedovanja velikega števila avtorjev iz nizkoresolucijskih slik. Predlagano rešitev ovrednotimo na tekmovanju spletnega portala Kaggle, kjer je za pare digitalnih umetnin potrebno napovedati, ali sta delo istega slikarja. V nalogi pokažemo, da značilke, izpeljane iz tem in motivov slik, podobno kot nižjenivojske značilke vsebujejo inherentne lastnosti, ki so primerne za razlikovanje med avtorji.

Ključne besede

strojno učenje, globoki modeli, konvolucijske nevronske mreže, umetniške slike

Abstract

An increasing number of studies are investigating how to automatically recognize painters from digital artwork images. We approach this problem in a supervised manner, by training a high-capacity convolutional neural network, capable of predicting a large number of artists from low-resolution scans. We evaluate the proposed solution in a Kaggle competition, in which pairs of paintings need to be classified based on the identity of their authors. The main contribution of our work is the provision of empirical evidence that themes and motifs, similar to low-level features, contain discriminative potential for identifying painters.

Keywords

machine learning, deep models, convolutional neural networks, artwork images

Razširjeni povzetek

Prepoznava pristnih slikarskih del med vseprisotnimi ponaredki je zaradi visokih cen umetnin izrednega pomena [1, 2]. Poleg tradicionalnih metod odkrivanja se v zadnjih letih uveljavljajo pristopi, ki slednje dopolnjujejo in temeljijo na računalniškem vidu ter statistični analizi tekstur in sledi čopiča. Uspešnost tovrstnih metod je v veliki meri odvisna od prisotnosti visokoresolucijskih slik, ki so nujno potrebne za izpeljavo nizkonivojskih značilk za učenje [3, 4]. Cilj našega dela je ovrednotenje tehnike globokega učenja, ki je sposobna napovedovanja velikega števila avtorjev iz nizkoresolucijskih slik in avtomatskega pridobivanja značilk, primernih za razlikovanje med njimi. Algoritem preizkusimo na tekmovanju Painter by Numbers[1] s spletnega portala Kaggle, kjer je za pare digitalnih umetnin potrebno napovedati, ali sta delo istega slikarja. Poleg tega pokažemo, kako znanje naučenega modela prenesti na reševanje sorodnih problemov. Funkcionalnost izpostavimo v obliki namestitve modela na oddaljeni strežnik, ki je dostopen iz odprtokodnega orodja za strojno učenje in vizualizacijo podatkov Orange [5].

Kljub temu, da so ostali udeleženci tekmovanja na Kaggleu uporabili sorodne metode, je naše delo, v kolikor nam je znano, edina objavljena raziskava, ki z uporabo globokega učenja razvršča slike v več kot 1.500 kategorij.

[1] `https://www.kaggle.com/c/painter-by-numbers`

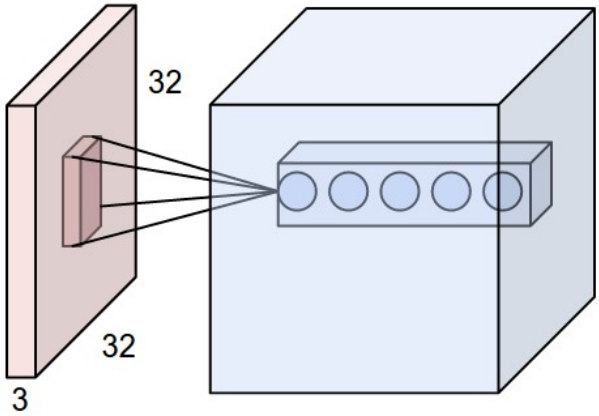
I Sorodna dela

Klasični pristopi algoritmične obdelave umetnin se nanašajo predvsem na klasifikacijo glede na avtorje in klasifikacijo glede na stilske lastnosti slik. V obširni študiji so S. Karayev in sod. [6] razvrščali slike glede na stil, tako da so nižjenivojskim statistikam dodali značilke, povezane s kompozicijo in vsebino. Večina sorodnih del, ki se nanaša neposredno na razpoznavanje slikarjev, procesira umetnine Vincenta van Gogha. C. R. Johnson in sod. [4] so s pomočjo nekaj raziskovalnih skupin izvedli izčrpno študijo identifikacije pristnih del van Gogha med znanimi ponaredki. J. Li in sod. [7] so razvili metodo za opis sledi čopiča in podali empiričen dokaz o edinstvenih potezah in stilu van Gogha. Novejše delo G. Folega in sod. [8] vsebuje javno dostopno podatkovno množico visokoresolucijskih slik, avtorji pa uporabijo konvolucijsko nevronsko mrežo za izpeljavo značilk, ki so uporabne za razlikovanje med pristnimi in ponarejenimi deli. V zvezi z ostalimi slikarji so S. Lyu in sod. [9] preverjali pristnost del Pietera Bruegela, L. Shamir in sod. [10] pa so preučevali avtomatsko klasifikacijo devetih umetnikov.

Poudariti je potrebno, da se sorodna dela v veliki meri osredotočijo na značilke, povezane z vizualnimi učinki tekstur, z obliko in usmerjenostjo sledi čopiča ter z barvami, kar povzema predvsem stilske značilnosti posameznih slikarjev. V nalogi raziščemo primernost gostih predstavitev slik, ki se v nasprotju s sorodnimi deli navezujejo na teme in motive ter so avtomatsko generirane s pomočjo globokega modela. S tem omogočimo uporabo večjih podatkovnih množic, sestavljenih iz slik poljubnih resolucij in gostot slikovnih točk.

Konvolucijska nevronska mreža je transformacija $f(x) \rightarrow s$, kjer je s k-dimenzionalni vektor, čigar elementi se seštejejo v ena in predstavljajo verjetnostno porazdelitev k-tih razredov, x pa je običajno 3-dimenzionalna RGB slika. Glavni koncepti tovrstnih mrež so lokalna povezanost, deljene uteži filtrov, zmanjševanje dimenzij in uporaba velikega števila nivojev [11].

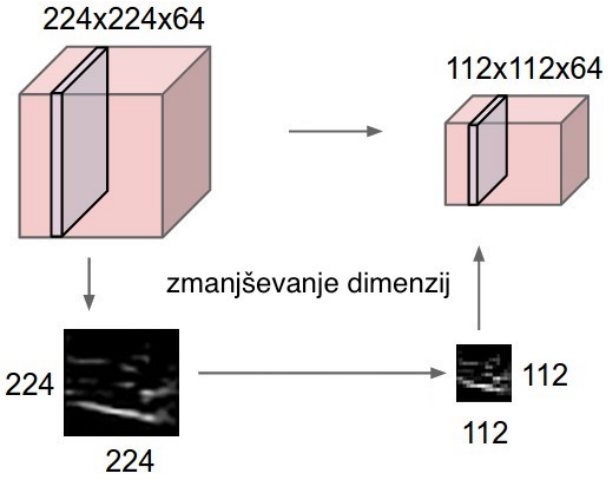
Y. LeCun in sod. [12] so v svojem delu z gradientno metodo optimizirali konvolucijske nivoje, ki so osnovni gradniki modela. Primer tovr-



Slika 1: Primer prvega konvolucijskega nivoja, ki kot vhod dobi RGB sliko velikosti 32×32 slikovnih točk prikazano z rdečim volumnom. Množica nevronov nivoja je prikazana z modro barvo. Vsak nevron je povezan z manjšim delom vhoda in se razteza čez celotno globino (trije barvni kanali). Pet nevronov, ki so povezani z istim delom slike, pripadajo različnim filtrom nivoja in skupaj tvorijo globino izhoda.[2]

stnega nivoja je prikazan na sliki 1. Sestavljen je iz množice filtrov, vsak izmed njih pa se razteza čez celotno globino vhodnega volumna in vsebuje $visina \times sirina \times globina$ uteži w, ki se spreminjajo med učenjem. Filtre premikamo v smeri dolžine in širine vhodnega volumna, pri čemer je velikost koraka konstantna, na vsakem mestu pa izračunamo skalarni produkt med trenutno pozicijo x in utežmi w. Izhodi različnih filtrov se kopičijo v globino, ki je natančno določena s številom filtrov v nivoju [12]. Dimenzije vhoda običajno ohranjamo z dodajanjem ničel na robove tenzorjev, s čimer obenem ohranjamo tudi informacijo. V nasprotju z lokalno povezanostjo konvolucijskega nivoja, ki je posledica majhnih vrednosti parametrov *visina* in *sirina*, polno povezan nivo vsebuje povezave vseh vhodov m z vsemi nevroni n in vsebuje $m \times n$ uteži.

Lokalna povezanost modelu omogoča pridobivanje osnovnih značilk, ki nato služijo kot vhod v naslednje konvolucijske nivoje in so posledično združene v vse bolj kompleksne oblike. Uteži enega filtra so identične, torej



Slika 2: Primer združevalnega nivoja, ki zmanjšuje dimenzije in deluje na vsakem sloju globine 224×224 dimenzionalnega vhodnega volumna posebej. V tem primeru nivo razpolovi število nevronov v smeri višine in širine, kar na izhodu tvori 112×112 dimenzionalni volumen z isto globino.[2]

deljene, na vseh pozicijah vhoda [12]. Zmanjševanje dimenzij podatkov dosežemo z dodajanjem združevalnih nivojev, ki jih podobno kot konvolucijske nivoje premikamo v smeri dolžine in širine vhodnega volumna. Slednji delujejo na vsakem sloju globine posebej, kot je prikazano na sliki 2. Z izbiro največjega elementa ali pa z izračunom povprečne vrednosti le-teh na vsaki poziciji združujemo semantično podobne značilke [12, 11].

Pomemben del nevronskih mrež so aktivacije, ki jih vstavimo za konvolucijskimi in polno povezanimi nivoji ter modelu omogočijo modeliranje nelinearnih funkcij. ReLU aktivacije, oblike $f(x) = \max(0, x)$ povečajo hitrost konvergence v primerjavi s sigmoidno in tanh funkcijo, ki sta se uporabljali v preteklosti [13, 14]. V delu uporabimo različico PReLU avtorjev K. He in sod. [15], oblike $f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x \geq 0)(x)$, kjer je α učljiv parameter. Izhodni nivo mreže je implementiran s posplošitvijo logistične funkcije oziroma s softmax funkcijo, ki slika prostor realnih k-dimenzionalnih vektorjev v prostor k-dimenzionalnih vektorjev, katerih elementi so vrednosti med

[2]Slika s spletne strani `http://cs231n.stanford.edu`.

0 in 1 ter se seštejejo v 1 [16].

Delo K. Simonyan in sod. [17] vsebuje dokaz o znatnem vplivu števila nivojev na učinkovitost konvolucijske nevronske mreže. Homogeno arhitekturo VGG, pri kateri enote iz dveh do treh konvolucijskih nivojev in združevalnega nivoja zlagamo enega za drugim, povzamemo v našem delu. Konvolucijski nivoji so sestavljeni iz filtrov, velikosti 3×3 in z velikostjo koraka 1, združevalni nivoji pa so velikosti 2×2 in imajo velikost koraka 2.

Vzvratno razširjanje napake je algoritem za računanje delnih odvodov sestavljene fukncije ali praktična implementacija verižnega pravila. Y. LeCun in sod. [18] so metodo prvi uporabili za učenje konvolucijskih nevronskih mrež. Vsak nivo mreže lahko razumemo kot posamezno funkcijo z možnimi vhodnimi parametri, ki jih optimiziramo med učenjem. V idealnem primeru poznamo uteži, ki maksimizirajo funkcijo verjetja ali aposteriorno verjetnost, v primeru kazni za uteži glede na dane učne podatke. Tako rešitev poiščemo z izračunom napake, ki je pri klasifikaciji običajno enaka negativnemu logaritmu funkcije verjetja, in z vzvratnim razširjanjem le-te vse do prvega nivoja. Po določitvi vseh lokalnih delnih odvodov lahko spremenimo parametre modela in v naslednji iteraciji pričakujemo manjšo napako.

A. Krogh in sod. [19] so potrdili, da L2 regularizacija zmanjša prekomerno prilagajanje nevronskih mrež učnim podatkom. Novejši pristop za regularizacijo globokih modelov so pred kratkim razvili N. Srivastava in sod. [20]. Pri tej metodi pred polno povezane nivoje na koncu mreže dodamo posebne nivoje, ki izhode nevronov med učenjem naključno izpustijo. Dodatno uporabimo nivoje za paketno normalizacijo avtorjev S. Ioffa in sod. [21], ki so vstavljeni po konvolucijskih in polno povezanih nivojih ter pred aktivacijskimi funkcijami ter modelu omogočijo centriranje in skaliranje posameznih paketov vhodnih podatkov.

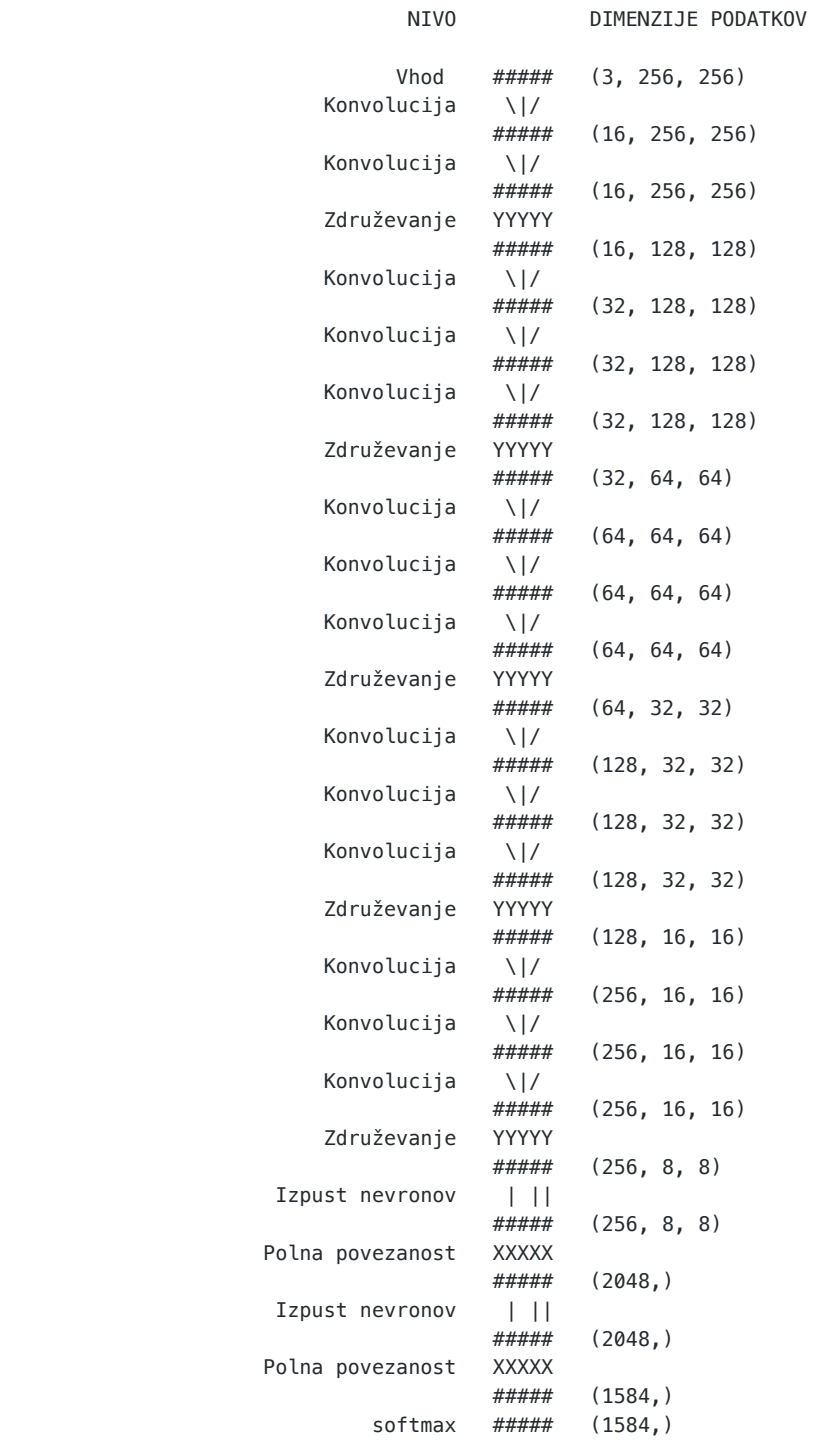
II Konvolucijske nevronske mreže avtorstva umetniških slik

V tem razdelku predlagamo konvolucijsko nevronsko mrežo, povzeto po arhitekturi VGG za klasifikacijo umetniških slik v enega izmed 1.584 razredov s tekmovanja Painter by Numbers.[3] Dodatno opišemo metodo učenja, razširitev algoritma za problem preverjanja, ali dve sliki pripadata istemu razredu, in končno rešitev tekmovanja.

Na sliki 3 je prikazana arhitektura najbolj uspešnega modela, za večjo razumljivost brez nelinearnosti in nivojev za paketno normalizacijo. Prekomerno prileganje učnim podatkom je dodatno omejeno z uporabo L2 kazni za uteži in nivoja za naključno izpuščanje nevronov, ki je prisoten le pred polno povezanimi nivoji na koncu mreže. Uporabljeni so konvolucijski filtri, velikosti 3×3 z velikostjo koraka 1. Po konvoluciji so na vsako stran tenzorja vrinjene ničle, kar ohranja prvotne dimenzije volumna. Dimenzije zmanjšujejo 2×2 združevalni nivoji z velikostjo koraka 2. Za nelinearnosti je uporabljena funkcija PReLU.

Model s slike 3 smo zaradi omejitev s pomnilnikom učili v paketih po 96 primerov, in sicer je za konvergenco potrebnih približno 280 iteracij. Parametre modela posodabljamo z algoritmom Adam s stopnjo učenja 7.4e-05 [22]. Celoten proces učenja traja približno štiri dni na grafični procesni enoti GeForce GTX TITAN X z 12 GB pomnilnika. Krajšo stranico vsake slike spremenimo na velikost 256 slikovnih točk, s čimer ohranimo razmerje stranic, nato pa jo obrežemo na sredini večje stranice in s tem dobimo sliko, velikosti 256×256 slikovnih točk. Predprocesiranje vključuje tudi centriranje in skaliranje posameznih slikovnih točk, s čimer zagotovimo vhodne podatke s povprečjem nič in enotsko varianco, kar je ugodna lastnost za gradientne metode optimizacije. Med samim učenjem vhodne slike naključno obračamo, približujemo, premikamo in raztegujemo, kar dodatno omeji prekomerno prileganje modela učnim primerom.

[3] `https://www.kaggle.com/c/painter-by-numbers`



Slika 3: Arhitektura najbolj uspešnega modela s štirinajstimi konvolucijskimi nivoji. Nelinearnosti in nivoji za paketno normalizacijo zaradi večje razumljivosti niso prikazani.

Metrike za preverjanje, ali dva primera pripadata istemu razredu, izhajajo predvsem s področja preverjanja obrazov in so neposredno uporabne za domeno umetniških slik. Pri nenadzorovanem načinu brez dodatnega učenja uporabimo model za napovedovanje enega izmed možnih razredov, tako da izračunamo skalarni produkt verjetnostnih porazdelitev k-tih razredov obeh slik (izhodov zadnjega nivoja mreže). Pri nadzorovanem načinu ali siamski arhitekturi model repliciramo za vsako sliko para, njuna izhoda pa združimo v vektor, ki ga povežemo s polno povezanim nivojem za napovedovanje verjetnosti istega razreda. V končni rešitvi je uporabljen nenadzorovan način, računanje skalarnega produkta obeh verjetnostnih vektorjev, predvsem zaradi časovnih omejitev in dobrih rezultatov na tekmovanju.

Naša najbolj uspešna rešitev na tekmovanju Painter by Numbers[4] s spletnega portala Kaggle je ansambel 18 konvolucijskih nevronskih mrež. Eno hipotezo smo iz množice modelov izpeljali z izračunom uteženega povprečja napovedi v enega izmed možnih razredov in z izračunom skalarnega produkta parov povprečnih vektorjev. Uteži smo izbrali glede na napako modela v validacijski množici. Mreže v ansamblu se razlikujejo v številu konvolucijskih nivojev, številu nivojev za izpuščanje nevronov, v L2 regularizaciji in številu iteracij učenja.

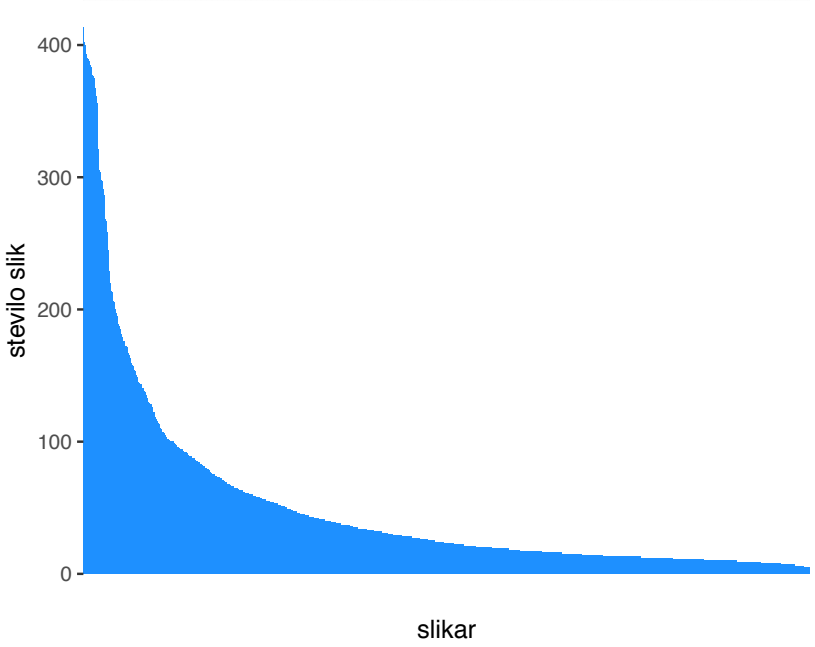
III Eksperimentalno ovrednotenje

V nadaljevanju podamo kratek opis javno dostopne podatkovne množice, definiramo ogrodje za validacijo modela in prikažemo končne rezultate s tekmovanja Painter by Numbers.[4]

Večina umetniških slik je s spletne strani WikiArt.[5] Podatkovna množica je neuravnovešena, kar je vidno na sliki 4. Učna množica je sestavljena iz 79.433 slik 1.584 slikarjev, v testni množici pa je skupno 23.817 umetnin avtorjev, katerih dela niso nujno prisotna v učni množici. Učna množica je

[4] `https://www.kaggle.com/c/painter-by-numbers`

[5] `https://www.wikiart.org`



Slika 4: Število slik vsakega umetnika v učni podatkovni množici.

dodatno razdeljena na pravo učno množico in validacijsko množico, na način ohranitve deleža slik avtorjev, s čimer dobimo 71.423 učnih primerov in 8.010 validacijskih primerov. Končna testna množica parov slik je sestavljena iz 23.817 testnih umetnin, ki so razdeljene v 13 skupin glede na stilske metapodatke s strani WikiArt,[5] napovedi pa je potrebno izračunati za vsak par slik znotraj posamezne skupine, kar skupno nanese približno 22 milijonov primerov. Običajno se na tovrstnih tekmovanjih tudi testna množica razdeli na dva dela, kar preprečuje prekomerno prileganje na javno dostopni lestvici. Končni rezultati so torej izračunani na približno tridesetih procentih 22 milijonov parov in sestavljajo skrito lestvico, ki postane javno dostopna šele po zaključku tekmovanja.

Rezultati prvih petih udeležencev tekmovanja so podani v tabeli 1 v obliki mere uspešnosti AUC. Prikazana je skrita lestvica s spletnega portala Kaggle, kjer je naš ansambelski pristop dosegel najvišji rezultat.

model	AUC testnih parov
1. mesto (naš ansambel)	**0.92890**
2. mesto	0.91929
3. mesto	0.90440
4. mesto	0.87763
5. mesto	0.83419

Tabela 1: Rezultati prvih petih udeležencev tekmovanja Painter by Numbers s spletnega portala Kaggle. Mera uspešnosti AUC je podana s skrite lestvice, kjer je naš ansambelski pristop dosegel najvišji rezultat.

IV Razprava

Zaželeni lastnosti splošnega orodja avtorstva umetniških slik sta uspešnost napovedovanja na delih še nevidenih slikarjev in razlikovanje med najbolj podobnimi avtorji. Ta dva vidika raziščemo z analizo rezultatov tekmovanja. Opišemo tudi način pridobitve gostih, semantičnih predstavitev poljubnih slik, ki jih preizkusimo na sorodnem problemu prepoznavanja slik Vincenta van Gogha.

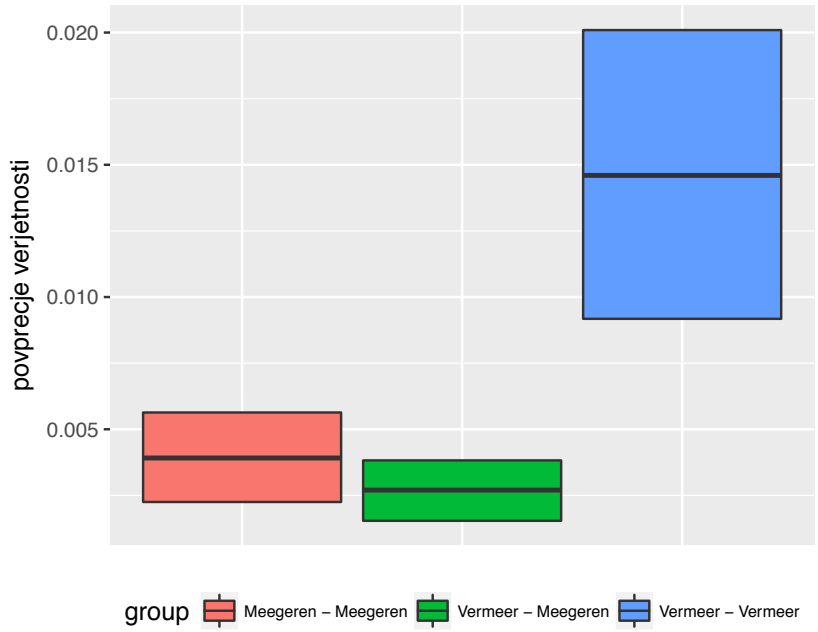
Ekstrapolacija ansambelskega pristopa je podana kot vrednost mere uspešnosti AUC za dve različni skupini napovedi iz testne množice parov. V prvi skupini so pari slik avtorjev, katerih dela so prisotna v učni množici, v drugi skupini pa so pari slik še nevidenih umetnikov. Rezultati so podani v tabeli 2, kjer je razvidno, da je napovedovanje na neznanih avtorjih manj uspešno, kar je domnevno posledica uporabe nenadzorovanega načina preverjanja, ali dva primera pripadata istemu razredu.

V testni množici parov so tudi dela znanega nizozemskega ponarejevalca Hana van Meegerena, ki je oponašal nekatere najbolj znane umetnike, med njimi tudi Johannesa Vermeera. Na desni strani slike 5 je primer umetnine, ki je dolgo veljala za pristno delo Vermeera, poleg njegove avtentične slike za primerjavo.

model	AUC videnih slikarjev	AUC nevidenih slikarjev
ansambel	0.9434	0.8256

Tabela 2: Mera uspešnosti AUC dveh različnih skupin napovedi za testne množice parov. V prvi skupini so pari slik avtorjev, katerih dela so prisotna v učni množici, v drugi skupini pa so pari slik še nevidenih umetnikov.

Slika 5: Primer podobnih slik Vermeera in van Meegerena. Johannes Vermeer: The Milkmaid (levo) in Han van Meegeren: Supper at Emmaus (desno). Sliko Supper at Emmaus so dolgo obravnavali, med drugimi tudi poznavalca Abraham Bredius and M. Jean Decoen, kot pristno delo Vermeera.[6]



Slika 6: 95% intervali zaupanja aposteriornih povprečij verjetnosti, da sta sliki delo istega avtorja. Pari so razdeljeni v skupine glede na to, ali je obe sliki ustvaril Vermeer, obe van Meegeren ali vsak eno.

V testni množici je na voljo 20 slik Hana van Meegerena in 13 slik Johannesa Vermeera oziroma 528 parov njunih del. S pomočjo MCMC vzorčenja [23] ocenimo 95% intervale zaupanja povprečja verjetnosti, da sliki pripadata istemu avtorju. Vizualna primerjava za pare Vermeera, pare van Meegerena in mešane pare je prikazana na sliki 6. Precej prepričani smo, da je povprečna verjetnost za pare dveh slik Vermeera višja kot za mešane pare, po drugi strani pa je povprečna vrednost slik van Meegerena bližje različnim parom. Ponarejevalec je res slikal v stilu različnih umetnikov, potrebno pa se je zavedati, da njegova dela niso prisotna v učni množici, kar je glavni razlog za nižje verjetnosti.

Konvolucijske nevronske mreže spreminjajo prostor vhodnih slik v nižjedimenzionalen prostor, v katerem so primeri linearno ločljivi. Izhod predza-

[6]Sliki s spletne strani `https://www.wikiart.org`.

dnjega nivoja mreže lahko torej uporabimo kot način avtomatskega izpeljevanja značilk. To funkcionalnost izpostavimo na oddaljenem strežniku, ki je dostopen iz odprtokodnega orodja za strojno učenje in vizualizacijo podatkov Orange [5].

G. Folego in sod. [8] v svojem delu objavijo tudi javno dostopno podatkovno množico 333 visokoresolucijskih slik van Gogha in njemu sorodnih avtorjev ter samo razdelitev na učni in testni del. Osredotočijo se na problem razlikovanja med pristnimi slikami in slikami, ki oponašajo stil umetnika. Njihov pristop vključuje razdelitev vsake slike na manjše dele, izpeljavo značilk posameznih delov s pomočjo naučene konvolucijske nevronske mreže in klasifikacijo ter združevanje v končno napoved. Značilke predzadnjega nivoja modela s 14 konvolucijskimi nivoji in *Inception-v3* modela [24] preizkusimo z učenjem logistične regresije na isti učni množici kot G. Folego in sod. S trikratnim prečnim preverjanjem na učni množici poiščemo hiperparametre modela in izračunamo napovedi za slike v testni množici. Rezultati so podani v tabeli 3. Značilke modela s 14 konvolucijskimi nivoji so le nekoliko boljše od značilk *Inception-v3* modela in vrednost F1 je za 0.029 točk nižja od najslabšega pristopa G. Folega in sod. Po drugi strani pa so značilke našega modela bolj uspešne z vidika števila imitiranih slik, ki so bile označene kot van Goghove. Matrika dejanskih razredov in napovedi modelov za oba razreda je prikazana v tabeli 4, kjer so van Goghove slike označene z *vg*, sorodne slike pa z *nvg*.

Predpostavimo, da je glavni razlog za nižjo vrednost F1 manjša količina informacije, ki jo izkoristi naš pristop. Uspešnost metode z vidika števila sorodnih slik, ki so bile označene kot van Goghove, pa kljub temu dokazuje, da so značilke, izpeljane iz nizkoresolucijskih slik, informativne za problem prepoznavanja avtorjev in jih je smiselno upoštevati pri napovedovanju.

model	AUC	CA	F1
model s 14 konv. nivoji	0.972	0.896	0.851
Inception-v3	0.909	0.881	0.840
G. Folego in sod.	-	-	0.880, 0.906, 0.923

Tabela 3: Mera uspešnosti AUC in napovedna točnost logistične regresije, naučene na značilkah modela s 14 konvolucijskimi nivoji in *Inception-v3* modela za razlikovanje med van Goghovimi in sorodnimi slikami. Mera uspešnosti F1 je podana za primerjavo z metodo G. Folega in sod.

model s 14 konv. nivoji

dejanski razred	vg	nvg
vg	20	5
nvg	2	40

Inception-v3

	vg	nvg
vg	21	4
nvg	4	38

G. Folego in sod.

	vg	nvg
vg	24	1
nvg	3	39

Tabela 4: Matrika dejanskih razredov in napovedi logistične regresije, naučene na značilkah modela s 14 konvolucijskimi nivoji in *Inception-v3* modela za razlikovanje med van Goghovimi in sorodnimi slikami. Za primerjavo je dodana matrika najbolj uspešnega pristopa G. Folega in sod. Van Goghove slike so označene z *vg*, sorodne slike pa z *nvg*.

V Sklep

Avtomatska prepoznava originalnih umetniških del med ponaredki je odprt raziskovalni problem. V nalogi smo razvili splošno orodje za napovedovanje velikega števila slikarjev iz digitalnih slik in prikazali primernost uporabe le tega za reševanje sorodnih problemov. Namerna izpustitev nizkonivojskih značilk nam je omogočila učenje iz večje podatkovne množice, sestavljene iz nizkoresolucijskih slik.

Napovedne modele smo preizkusili na tekmovanju spletnega portala Kaggle, kjer je bilo za pare slik potrebno napovedati, ali sta delo istega slikarja. Z ansambelskim pristopom smo dosegli vrednost 0.92890 mere uspešnosti AUC in s tem zasedli prvo mesto. Dodatno smo preizkusili delovanje rešitve na sorodnih problemih in pokazali, da značilke, izpeljane iz tem in motivov slik, vsebujejo lastnosti, ki so primerne za razlikovanje med avtorji.

Kljub vzpodbudnim rezultatom vsebuje naš pristop prostor za izboljšave. Menimo, da bi uporaba nadzorovanega načina za preverjanje, ali dva primera pripadata istemu razredu, pripomogla k boljšemu rezultatu pri klasifikaciji parov. Poleg tega bi najsodobnejše arhitekture konvolucijskih nevronskih mrež v povezavi z uporabo naučenih modelov na večjih množicah podatkov prav tako izboljšale rezultate. Naše delo vseeno zastavlja pomembno vprašanje: na kakšen način združiti informacijo nizkonivojskih značilnosti in semantičnih značilk, v primerih, ko so strokovnjakom na voljo visokoresolucijske slike umetnin.

Chapter 1

Introduction

Identification of authentic paintings among ubiquitous forgeries is a matter of the utmost importance given the price tags of genuine, highest reputation artwork [1, 2]. Traditionally, connoisseurs resorted to a chemical or physical analysis of materials, inspection of work's provenance and to visual examination. In the last ten years, computer vision researchers have started to collaborate with museums and art scholars to develop techniques that would supplement visual investigation and aid the process of determining the authenticity of paintings [25, 3]. The success of these approaches in many cases relies on analysis of detailed features such as textures and brushstrokes, which requires high-resolution digital scans of original paintings [3, 4]. The focus of this work, however, is an assessment of a utility of deep learning that extracts features for predicting authors directly from low-resolution artwork images.

The process of building and validating our technique for authorship prediction was carried out within a Kaggle competition called Painter by Numbers.[1] The challenge of the competition was to examine pairs of paintings to determine if they were painted by the same artist. An example of a dissimilar pair of artworks by Albrecht Dürer, a painter and printmaker of the German Renaissance, is given in Figure 1.1. The artist mainly utilized woodcut, a relief printing technique, but also produced some watercolours and oil

[1] `https://www.kaggle.com/c/painter-by-numbers`

Figure 1.1: An example of different paintings by the same painter. Albrecht Dürer: Willow Mill, 1496 - 1498 (left) and Albrecht Dürer: The Great Courier, 1495 (right).[2]

Figure 1.2: An example of similar paintings by different painters. Claude Monet: The Grenouillère, 1869 (left) and Pierre-Auguste Renoir: La Grenouillere, 1869 (right).[2]

paintings, resulting in a heterogeneous collection of works. Figure 1.2 gives a contrary example of a similar pair of artworks by different artists, albeit both French and leading painters of Impressionism. These two examples argue for a high level of intricacy of the problem being solved, but the reader should also be aware of somewhat more facile examples, like the one in Figure 1.3, where both paintings are similar and painted by the same artist.

In this work, we propose a deep convolutional neural network that is capable of predicting a large number of artists from digitized paintings, and

[2]Images from `https://www.wikiart.org`.

Figure 1.3: An example of similar paintings by the same painter. Vincent van Gogh: The Starry Night, 1889 (left) and Vincent van Gogh: Red Vineyards at Arles, 1888 (right).[2]

show its applicability for the task of classifying pairs of images based on their authors, as provided in the Kaggle competition. Additionally, we show how the model could further be exploited by exerting transfer learning, a technique of reusing knowledge from one task to solve another [26, 27], to compare results to a recent work of identifying Vincent van Gogh's paintings among related, non-van Goghs [8]. Furthermore, we deploy the trained neural network to a remotely accessible service that exposes the transfer learning functionality and depict its usage in the Orange open source machine learning and data visualization software [5].

Notwithstanding that other participants in the Kaggle competition have used similar approaches, to the best of our knowledge, this work is the only published research using deep learning to classify artwork images into more than 1,500 categories.

Chapter 2

Related Work

Applications in the field of digital image processing are entering more narrow domains like that of computerized processing of paintings, establishing an interdisciplinary field of computer vision and visual arts [25]. An increasing number of researchers are investigating how to automatically identify authors from paintings [3, 8]. In the last few years, human-level performance has been achieved, and even surpassed in some domains, in the field of visual object recognition by utilizing deep learning [11].

2.1 Classical Processing of Artwork Images

Classical processing of digital artwork images primarily focuses on two tasks: classification of paintings with respect to their creators and classification based on art styles. We give an overview of both areas due to a high correspondence of techniques used to solve each of the problems, but it should be noted that only authorship classification attempts to solve a problem comparable to our work. A brief introduction to the general literature of computer vision applied in the study of visual arts, and with emphasis on analysing digitized paintings and drawings, is given in the paper by D. G. Stork [25].

Extensive research was performed for the task of classifying artwork images into genre, rather than author-based categories. J. Zujovic et al. [28]

explored steerable filter decomposition, canny edge detectors, and colour processing to extract texture, edge and colour features for classification. In a similar study, R. S. Arora et al. [29] focused on intermediate-level features such as line styles, geometry, perspective and classeme descriptors – semantic-level features, produced by a pre-trained classifier. In a comprehensive study, S. Karayev et al. [6] provided an evaluation of classifiers trained on several different features, including low-level statistics, colours, composition, and content. Y. Bar et al. [30] combined binary code representation of images and features from a pre-trained convolutional neural network to identify styles of paintings.

Most of the related research efforts for the author based classification task focused on investigating the works of Vincent van Gogh. C. R. Johnson et al. [4] conducted a comprehensive study, involving several research groups, of identifying authentic van Goghs among forgeries. The groups were working with 101 high-resolution gray-scale scans of paintings and analysed different wavelet templates to construct features for supervised learning. An overview of the same work and its extension is given in the paper by B. Cornelis et al. [3]. I. Berezhnoy et al. [31] analysed manipulation of complementary colours with respect to creation period of van Gogh's paintings. J. Li et al. [7] developed a method for extracting brushstrokes and produced scientific evidence of van Gogh's unique brushstroke styles. In a more recent study, G. Folego et al. [8] collected a public dataset of 333 high-resolution, density-normalized images, and employed a pre-trained convolutional neural network for extracting features that are useful for identifying van Gogh's paintings.

Regarding other painters, S. Lyu et al. [9] performed wavelet-like decomposition of high-resolution images to authenticate Pieter Bruegel's drawings and L. Shamir et al. [10] studied automated recognition of nine artists based on Fourier, wavelet, Chebyshev and other transforms for producing informative image features. E. Cetinic et al. [32] used intensity features from grayscale scans, colour and texture features to train classifiers on a collection of 500 paintings from 20 different artists. T. Mensink et al. [33] introduced

a sizable, public dataset comprised of digitized paintings among other art objects, and conducted baseline experiments for the artist classification task. F. S. Khan et al. [34] proposed a dataset consisting of artwork images from 91 different painters and demonstrated how local and global visual features performed for the artist and style classification tasks.

The reader should be aware of divergence between our proposed method and related work and should discern what key advantages and disadvantages are. Previous studies predominantly focus on low-level features that impart the notion of artist's personal style. These include, but are not limited to, visual effects of textures, shape and orientation of brushstrokes, and detailed manipulation of colours. This work, on the other hand, explores the amount of information encompassed in semantic image features that capture distinct themes and motifs of a painter, and are automatically devised by a convolutional neural network (ConvNet). An obvious disadvantage is that low-level characteristics that were shown to hold discriminative capabilities are not considered in this framework. Nonetheless, ConvNet-based approaches enable the exploitation of larger datasets comprised of low-resolution images, and eliminate an inconvenient dependency of high-resolution scans available in limited quantities [8].

It should also be noted that our approach attempts to solve a more general problem than related work. We do not consider the challenge of distinguishing between two specific artists, but rather aim to recognize a sizeable number of painters and show how this can further be utilized to answer more narrow questions. Another evident difference with some of the related studies is that we do not take pixel density (e.g. pixels per painted inch) into account. As a consequence, a classification bias might be introduced in our model, but we deem this necessary in order to retain the quality of being able to operate with a large number of training examples [4, 8, 7].

2.2 Processing of Artwork Images with Deep Learning

Recently, the research in image processing has focused on designing deep learning models that are able to generate digital artwork images of high perceptual quality. The first paper in the field was introduced by L. A. Gatys et al. [35] and later improved by J. Johnson et al. [36], V. Dumoulin et al. [37], L. A. Gatys et al. [38] and D. Ulyanov et al. [39]. These systems separate and recombine content and style of arbitrary images by extracting neural representations from features provided by pretrained neural networks. An aforementioned operation can transform photographies into works of art by blending their content with the style of a selected artistic image. Interestingly, there are already practical applications implementing this functionality.[1] Despite solving a different problem and not being directly related to our work, these studies provide an advance in algorithmic comprehension of how art is created and perceived [35].

2.3 Convolutional Neural Networks

Feedforward neural networks are trained to map fixed-size inputs to fixed-size outputs [11]. Formally, a convolutional neural network for image classification is a transformation $f(x) \rightarrow s$, where s is a k-dimensional simplex, representing a probability mass function over k possible classes, and x is usually a 3-dimensional RGB image. Four crucial concepts behind ConvNets are local connectivity, shared filter weights, downsampling and utilization of numerous layers [11].

A convolutional layer, an elementary unit of a ConvNet shown in Figure 2.1, was first trained with gradient based method by Y. LeCun et al. [12]. It consists of multiple learnable filters with small spatial dimensions (i.e. *height* and *width* or receptive field of the filter) that always extend to

[1] `https://deepart.io`

a full depth of the input volume [17]. Each filter is essentially a set of $height \times width \times depth$ weights w that are adjusted during the learning phase. All filters perform the same operation—a dot product between input x and w—on different locations of the input volume. Convolution can thus be formulated as shifting a filter along input's height and width and producing a single output at every possible position. Stride, zero-padding, and number of filters are hyperparameters that, together with the receptive field, affect the size of the output volume. Stride specifies the number of input positions for which each filter is shifted after calculating the dot product. For example, a $10\times10\times width$ input and a $3\times3\times width$ filter with stride 1 would produce an $8\times8\times1$ output. Zero-padding determines how many zeros are inserted around the borders of the volume and allows for retention of spatial dimensions of the input. It is also eminent for conserving information at the edges. The depth of the output corresponds directly to the number of filters in the processing convolutional layer [12]. In contrast to local receptive fields of convolutional layers, fully-connected (dense) layers connect each input with every output neuron in the layer, resulting in $m \times n$ weights, where m denotes the number of input neurons and n number of neurons in a processing dense layer. The output of a fully-connected layer is thus an n-dimensional vector.

Local connectivity is a direct consequence of small spatial dimensions of filters and enables extraction of fundamental image features such as oriented edges. These are then combined together into higher-level shapes by succeeding convolutional layers. Filter weights are shared in the sense that they are identical at every position of the input during convolution. The reasoning behind this concept is that feature extractors that are useful in one part of the image, are likely to be useful in other parts. Different feature detectors are thus learned by using multiple filters in a single layer [12].

Downsampling of the processed volumes is achieved by inserting pooling layers, shown in Figure 2.2, which combine together semantically similar features. Pooling layers, similar to convolutional layers, also incorporate notions of spatial extent and stride. Unlike convolutional layers, they operate

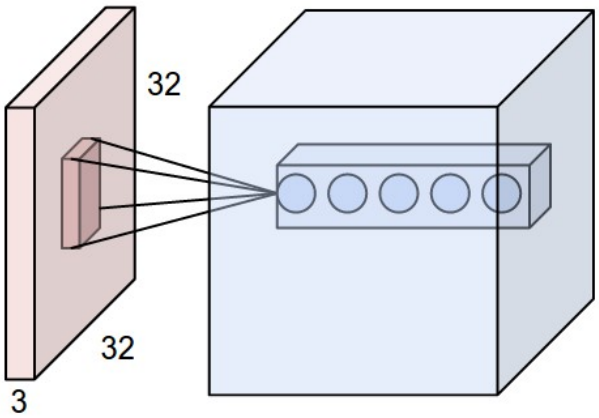


Figure 2.1: An example of a first convolutional layer whose input is a 32×32 RGB image, shown in red. Set of neurons in the layer is shown in blue. Each of them is connected to a local region of the input with its full depth (the three colour channels in this example). Five neurons that connect to the same region in the image belong to different filters in the layer and constitute the depth of the output volume.[2]

on each depth slice of the input volume separately and perform different calculations. Most common operations are selecting the maximum element or computing the average at each position. The motivation for their usage is that discovering features, and knowing their relative position to other features, is more important than storing their exact location [12, 11].

Another key notion of ConvNets are activation functions (non-linearities) applied to outputs of convolutional and dense layers. They enable neural networks to approximate non-linear functions (avert predictions to be linear transformations of the input). Rectified Linear Unit (ReLU) activations of the form $f(x) = \max(0, x)$, were shown to increase the speed of convergence of stochastic gradient descent compared to historically used sigmoid and tanh functions [13, 14]. A. L. Maas et al. [40] explored Leaky ReLU units of the form $f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x \geq 0)(x)$, where α is small and fixed, and showed their ability to avoid the zero gradient problem of ReLU activations, without having a significant impact on performance. K. He et al. [15] intro-

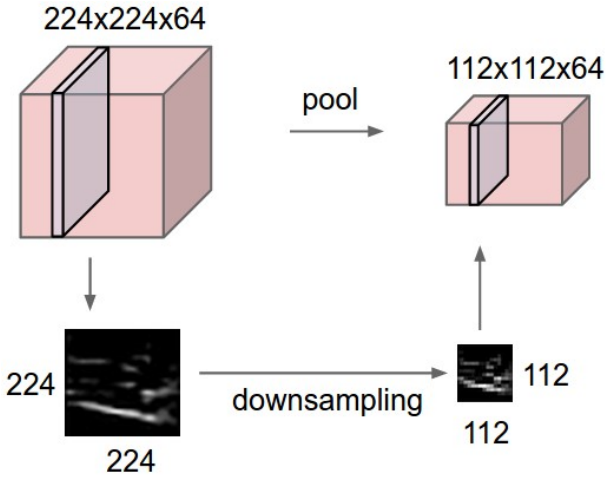


Figure 2.2: An example of a pooling layer that operates on each depth slice of a 224×224 dimensional input volume individually. In this example number of neurons is halved along height and width, resulting in 112×112 dimensional output.[1]

duced a PReLU function by turning α into a trainable parameter. Output layer of a neural network for classification is implemented with the softmax activation function, a generalization of the logistic function, that maps a space of real-valued k-dimensional vectors to a space of k-dimensional simplexes [16]. Simplexes consist of values between 0 and 1 that sum to 1, and can thus be interpreted as a probability distribution over k possible classes.

The main contribution of work by K. Simonyan et al. [17] is dispensing evidence that depth of a convolutional neural network has a substantial impact on performance. They proposed a homogeneous architecture known as VGG in which two to three convolutional layers are stacked together and followed by a pooling layer. All convolutional layers include 3×3 filters with stride 1 and zero-padding 1. Downsampling layers implement max pooling with spatial dimensions 2×2 and stride 2 to halve the number of neurons along each of the two dimensions. Their best-performing network consists of 16 convolutional layers and 3 fully-connected layers at the end of the network.

[1]Image from `http://cs231n.stanford.edu`.

The first paper on convolutional neural networks trained by backpropagation is by Y. LeCun et al. [18]. Backpropagation algorithm is a method for calculating derivatives of a function composition with respect to its input and is altogether a practical implementation of the chain rule [11]. Each layer of a neural network is essentially a separate function with possible trainable weights as inputs, joined together to form a computational graph. Ideally, we are interested in a set of weights that maximize the likelihood function, or a posteriori probability in case of weight penalties, given training images. To achieve that, the negative log-likelihood (or cross-entropy) loss function is calculated at the output of the network, and error term is propagated all the way back to the first layer through the application of chain rule. When all layers' local gradients are determined, the weights can be adjusted to produce a lower error rate at the next iteration (epoch). Training can thus be summed up as successive iterations of forward and backward passes, with the bjective of maximizing the cross-entropy function. It should be noted that the cost function has multiple local minima and therefore poses a high-dimensional, non-convex optimisation problem. A lot of researchers believe, however, that reaching local minima is adequate since their loss values are approximately similar, and that they approach the value of the global minimum as the network grows more complex [41].

A. Krogh et al. [19] theoretically and experimentally analysed effects of weight decay in feedforward neural networks and confirmed that better generalization can be achieved with L2 regularization. Model complexity is decreased with introduction of a weight penalty term into the cost function, which takes the following form: $E(w) = E_0(w) + \frac{\lambda}{2}\sum_i w_i^2$, where E_0 is an arbitrary error function and λ controls the strength of the regularization.

In a recent work, N. Srivastava et al. [20] developed a regularization technique called dropout that leads to significantly lower generalization errors. Every activation (neuron) in a layer remains active with some probability p during training, where p is a hyperparameter, and all other neurons are set to zero. When making predictions, dropout is disabled, but outputs need to

be scaled by p to compensate for smaller values expected by neurons in a successive layer. Each learning iteration with dropout is equivalent to sampling and training a smaller model from the possible 2^n networks, where n is a number of units in the original network and each of them either is or is not part of the sampled model. All sampled networks share a large number of weights. Test time is then an approximate averaging method of the 2^n trained models.

S. Ioffe et al. [21] recently developed a method that introduces the ability to preprocess mini-batches of training data at every layer of the network by forcing activations to take a unit Gaussian distribution. During training, a normalized version $\widehat{X}$ of mini-batch X is calculated by subtracting the mean and dividing by its standard deviation. Output is then defined as $\gamma\widehat{X} + \beta$, where γ and β are learnable parameters. Batch normalization has the capacity to represent the identity transformation, if proven useful, by setting γ to the value of variance and β to the expected value. Technique allows for higher learning rates, puts less emphasis on weight initialization and also has a regularizing effect.

2.4 Markov Chain Monte Carlo

A family of algorithms known as Markov chain Monte Carlo (MCMC) is often used for calculating numerical approximations of integrals in high dimensional spaces. In this work, we utilize MCMC for estimating confidence intervals of predictions for different groups of paintings. In Bayesian statistics, they are primarily employed for obtaining a sample of parameters of a model, particularly when the posterior distribution is not known (i.e. it is not clear how to draw samples from it and can only be evaluated up to a normalising constant). The main idea is to construct a Markov chain that spends more time in salient regions and imitates samples from a target distribution. This is achieved with the utilization of a random walk (e.g. Metropolis–Hastings algorithm, Gibbs sampling) or by exploiting information about the gradient

of the distribution (e.g. Hybrid Monte Carlo) [42]. Practical implementation of a No-U-Turn sampler, a variant of the Hybrid Monte Carlo algorithm, is available in the Stan probabilistic programming language [23].

Chapter 3

Convolutional Neural Networks for Painting Authentication

In 2012 A. Krizhevsky et al. [14] achieved state-of-the-art results in the ImageNet Large Scale Visual Recognition Challenge for image classification by training a deep convolutional neural network on a labeled dataset, composed of 1.2 million images from 1,000 classes. In this chapter we propose a ConvNet based on VGG architecture by K. Simonyan et al. [17] for classifying artwork images into 1,584 categories corresponding to 1,584 distinct painters, included in the Painter by Numbers competition.[1] We further describe the training method, techniques for identifying whether two input instances belong to the same class and our final solution for the competition.

3.1 Architecture

Figure 3.1 depicts the architecture of our single, best-performing model with non-linearities and batch normalization layers omitted for coherence. Batch normalization layers are added after every convolutional and dense layer, and before activation functions. In addition to that, L2 weight penalties are used, and dropout layers inserted before fully-connected layers to alleviate overfit-

[1] `https://www.kaggle.com/c/painter-by-numbers`

ting to the training set. Convolutional filters with 3×3 spatial dimensions and stride 1 are applied throughout the network to produce outputs that are two neurons smaller along each of the two dimensions, compared to the matching input volumes. Zero padding is used to retain the original shape after convolution and 2×2 max pooling with stride 2 halves the number of neurons along height and width. Dimensions of data flowing through the network can also be seen in Figure 3.1. PReLU activation was selected as a non-linear function. Keras[2] code of the described architecture is available in Appendix A.

3.2 Training Procedure

We trained the network from Figure 3.1 in batches of 96 examples due to memory constraints, and approximately 280 epochs were needed for the model to converge to local minima using the Adam optimizer with 7.4e-05 learning rate [22]. The learning process took roughly four days on a single GeForce GTX TITAN X GPU with 12 GB of RAM. The first preprocessing step was to resize each image's smallest dimension to 256 pixels (retaining the aspect ratio) and then cropping it at the center of the larger dimension, obtaining 256×256 images. Some information is lost during this process and an alternative approach where multiple crops are taken from the same image was considered, but not used for the final solution due to longer training times. Furthermore, mean values were subtracted from each feature in the data and the obtained values were normalized by dividing each dimension by its standard deviation. Preprocessing data statistics were computed from the subset of training instances. Random, label-preserving transformations such as rotations, zooms, shifts, shears, and flips were applied to images during the training phase. Data augmentation assures that ConvNet only rarely receives exactly the same data point more than once, and ergo further reduces overfitting. A significant implementation detail is that data augmentation was

[2]Deep Learning library available on `https://keras.io`.

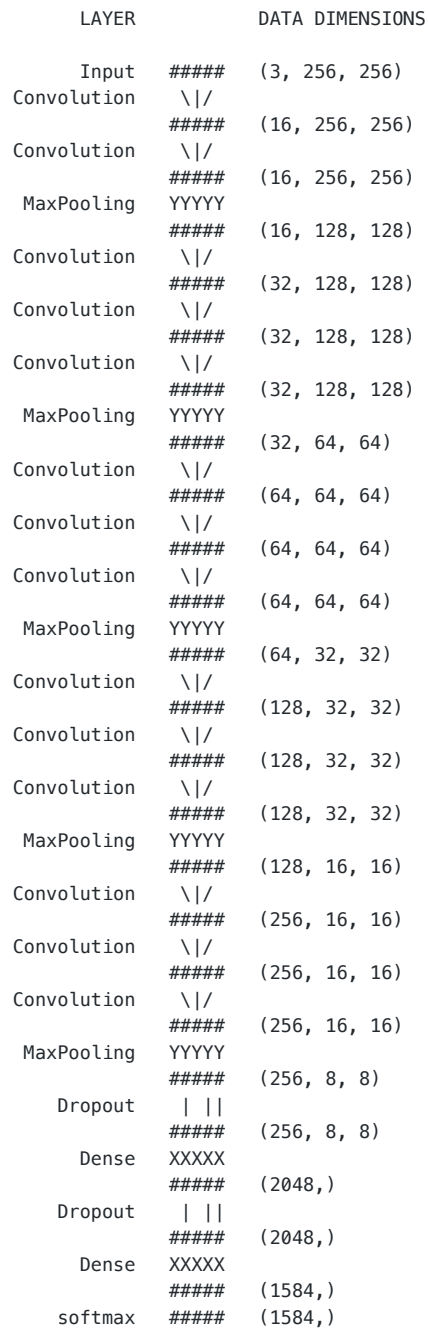


Figure 3.1: The architecture of the best performing neural network with fourteen convolutional layers. Non-linearities and batch normalization layers are omitted for coherence.

performed on new batches on CPU, at the same time as previous batches were processed on a GPU, not having any impact on training times [14].

3.3 Same-Class Verification

Same-class verification metrics have been a subject of extensive research in applications of face verification, and methods developed in this domain are directly applicable to the problem of determining whether two artwork images were created by the same painter [43, 44]. The unsupervised technique includes training a model that can predict one of the 1,584 classes and then calculating a dot product of the two class distribution vectors (i.e. outputs of the softmax layer). This approach can be described as treating the events A_i and B_i that two images are both of the i-th class as independent, allowing to multiply their probabilities, and then calculating the probability of all mutually exclusive events $P(A_i \cap B_i)$, where $i = 0, ..1583$, using the addition rule. The supervised method is an end-to-end metric learning approach called siamese network and is depicted in Figure 3.2. The main idea is to replicate the model once for each input image and merge their outputs into a single vector that can then be directly used to predict whether the two images were painted by the same artist. An important aspect of this architecture is that the weights of both models are shared in the sense that they are identical, and during backpropagation, the total gradient with respect to weights is the sum of the gradients contributed by the two models. The network trained for the unsupervised technique can also be used in the siamese architecture as an initialization approach followed by fine-tuning, a strategy of continuing the backpropagation for a related task, usually with a lower learning rate.

In spite of the fact that supervised approaches clearly outperform the unsupervised ones [44], we put most of the effort into the multi-class painter recognition task. For the final solution, the calculation of the dot product was used, mainly because it allowed us to invest more time in a thorough validation of the model, and achieved strong results in the competition.

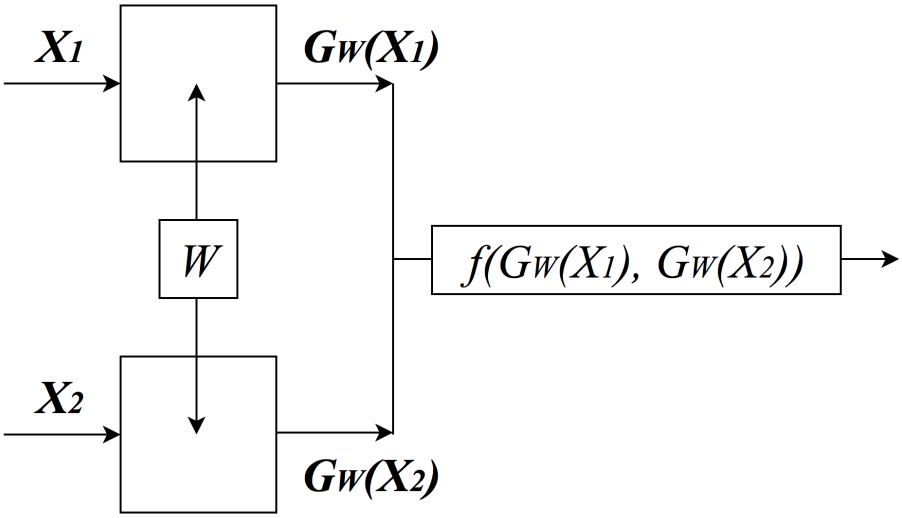


Figure 3.2: Siamese architecture (end-to-end learning verification metric). The weights W are shared between the two networks. Outputs $G_W(X_1)$ and $G_W(X_2)$ of both models are merged into a single vector, that is used to predict whether input images X_1 and X_2 are of the same class.

3.4 Competition Solution

Our overall best-performing solution in the Painter by Numbers competition[3] on Kaggle is an ensemble of 18 ConvNets trained during a hyperparameter search. All individual elements are shown in Table 3.1. Single hypothesis was derived from multiple models as a weighted average of their predictions for the multi-class painter recognition task, and then calculating dot products of pairs of averaged class distribution vectors. The weights for averaging were determined manually based on corresponding model's performance in the validation set. ConvNets in the ensemble differed in number of convolutional layers, number of dropout layers and L2 regularization. Moreover, early stopping, an approach in which network is saved at different epochs, was used to obtain distinct predictive models.

[3] `https://www.kaggle.com/c/painter-by-numbers`

conv layers	dropout layers	L2	epochs	validation log loss
14	2	0.003	279	2.9152
14	2	0.003	249	2.9162
14	2	0.004	305	2.9177
14	2	0.003	297	2.9244
14	2	0.003	269	2.9364
14	2	0.003	192	2.9420
14	2	0.003	207	2.9430
14	2	0.003	177	2.9538
11	2	0.004	201	2.9585
14	2	0.004	228	2.9595
14	2	0.003	220	2.9597
11	2	0.003	236	2.9631
11	2	0.004	241	2.9709
11	3	0.005	420	2.9904
11	2	0.002	305	2.9940
11	3	0.005	350	3.0083
11	2	0.002	267	3.0318
14	4	0.005	350	3.0510

Table 3.1: 18 ConvNets that constitute the final ensembled model used in the Painter by Numbers Competition on Kaggle. First two columns represent the number of convolutional and dropout layers, followed by L2 regularization strength, number of training epochs, and cross-entropy loss from the multi-class painter recognition task.

Chapter 4

Evaluation

In this chapter we give a brief description of the public dataset and define the evaluation framework used in the Painter by Numbers competition.[1] In the last section, we present results from the final, private leaderboard.

4.1 Dataset

The preponderance of images in the dataset are from WikiArt.[2] Number of artworks with respect to all painters in the training set is shown in Figure 4.1, where it can be seen that dataset is unbalanced. Moreover, minimum and maximum number of paintings by a single artist is 4 and 413, respectively. Overall, there are 79,433 images belonging to 1,584 painters in the training set, and the test set is comprised of 23,817 images, created by both, painters whose work is present in the training set, and additional artists not seen prior to test time.

[1] `https://www.kaggle.com/c/painter-by-numbers`
[2] `https://www.wikiart.org`

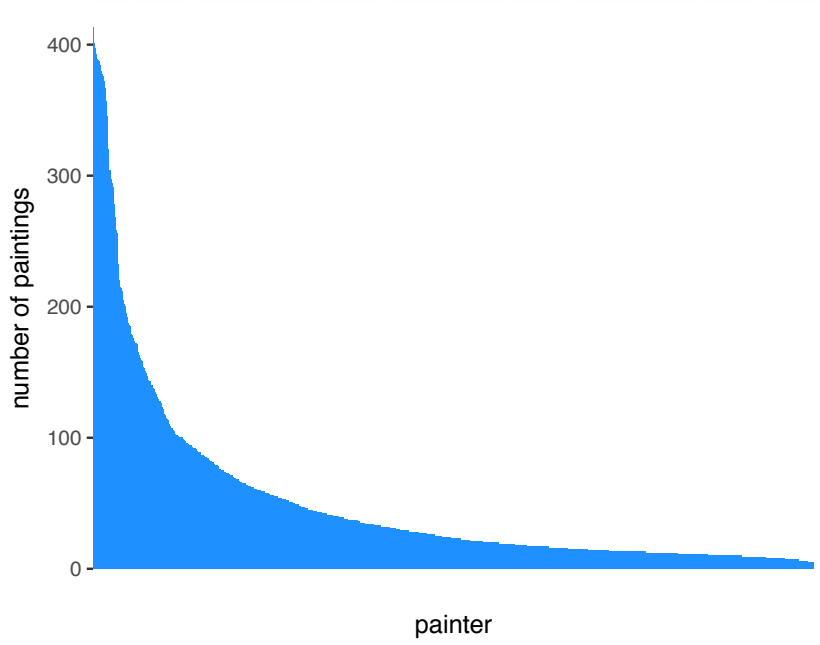


Figure 4.1: Number of paintings per each painter in the training set.

4.2 Evaluation Procedure

Predictive models were evaluated by splitting the entire dataset into three disjoint sets, with first partitioning into train and test groups already specified by the competition scheme. The training set was further separated into actual training and validation sets in a stratified manner, resulting in 71,423 training images and 8,010 validation images, both belonging to 1,584 classes. The final test set composed of pairs of paintings, for which predictions based on whether images are by the same artist are expected, was constructed from 23,817 images from the test set. The paintings were arranged into 13 groups in consonance with their artists' style metadata from WikiArt,[2] and all possible pairs within each group needed to be examined, producing approximately 22 million data points. The incentive behind the grouping procedure is that the algorithm is predominantly encouraged to discriminate between works of similar genres. It should be noted that in a typical Kaggle challenge there are two distinct leaderboards to prevent overfitting to the one that is publicly-

model	training log loss	validation log loss	test pairs AUC
8 conv layers	1.2811	3.0470	0.89300
11 conv layers	1.1277	2.9752	0.90123
14 conv layers	0.7061	2.9152	**0.90598**
17 conv layers	0.8285	3.0163	0.89454

Table 4.1: Results of our individual models. Cross-entropy loss from the multi-class painter recognition task is reported for training and validation sets. AUC is reported from the private leaderboard of the Painter by Numbers competition on Kaggle.

available during the competition, and that final scores were calculated on a private leaderboard only once after submissions had been closed. Test pair results reported in the next section were thus computed on approximately 30% of 22 million pairs.

4.3 Results

In Table 4.1 results of our individual models are given. Cross-entropy loss from the multi-class painter recognition task is reported for training and validation sets. AUC is reported from a private leaderboard on Kaggle. The architecture of the best performing single ConvNet labeled *14 conv layers* is depicted in Figure 3.1 in Chapter 3. Architectures and Keras[3] code of our remaining individual models, all with hyperparameter values identical to the *14 conv layers* network, are available in Appendix A.

In Table 4.2 results of first five participants from the competition are given. Again, AUC is reported from a private leaderboard on Kaggle, on which our ensembled approach achieved the highest score.

[3]Deep Learning library available on `https://keras.io`.

model	test pairs AUC
1st place (our ensemble)	**0.92890**
2nd place participant	0.91929
3rd place participant	0.90440
4th place participant	0.87763
5th place participant	0.83419

Table 4.2: Results of first five participants from the Painter by Numbers competition on Kaggle. AUC is reported from the private leaderboard, on which our ensembled approach achieved the highest score.

Chapter 5

Discussion

Generalizing to unknown artists and distinguishing between the most homologous painters are foremost attributes of a universal painting authorship tool. These aspects of the proposed solution are investigated by further analysing the results of the competition. In addition to that, we provide a straightforward application for obtaining dense, semantic features of arbitrary images, and demonstrate its practicality in two artist recognition tasks.

5.1 Extrapolation

Extrapolation of our ensembled approach to artists that were not seen during training is given in the form of AUC for two different groups of instances from the test set. The first group consists of pairs of images whose painters were present in the training set and the second group is composed of pairs whose artists haven't been seen during training. The results presented in Table 5.1 indicate that making predictions on unknown artists hurt the overall performance. We suspect that this is largely due to the fact that same class identity verification is calculated directly from the two class distribution vectors, and that an end-to-end supervised method (e.g. a siamese architecture) would manifest superior performance.

Visually comparing the mean probability of groups of seen and unseen

model	seen artists AUC	unseen artists AUC
ensemble	0.9434	0.8256

Table 5.1: AUC of our ensembled approach for two groups of instances from the test set. The first group consists of pairs of images whose painters were present in the training set and the second group is composed of pairs whose artists haven't been seen during training.

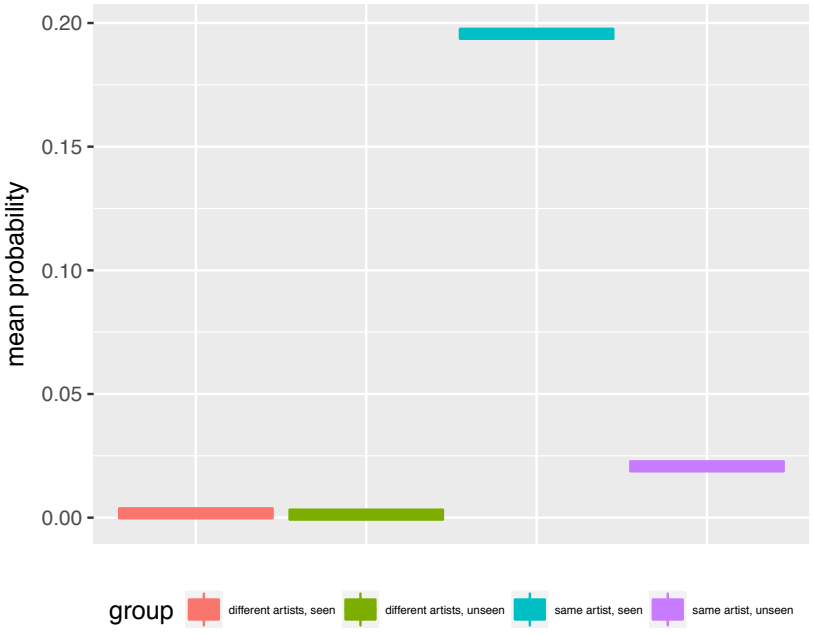


Figure 5.1: Posterior means of probability that two paintings are by the same artist. Pairs are split into groups based on whether painters were present in the training set (seen) or not present in the training set (unseen).

artists reveals additional insights into behaviour of the algorithm. We use a normal sampling model with uninformative priors but are only interested in posterior distribution of the mean, which we despite conjugacy, obtain via an MCMC sampler [23]. Diagnostic traceplots and sampling summaries are available in Appendix B, where the reader can convince herself that nothing anomalous can be spotted and that seemingly, all chains have converged.

Point estimates of posterior means of probability that two paintings are by the same artist are displayed in Figure 5.1. Confidence intervals are excluded since they do not provide any additional information due to large sample sizes. The plot confirms substantial discrepancy between predictions for seen and unseen painters where both images are by the same author and implies that mean prediction for pairs with same, unseen painter is closer to those of different artists.

Posterior means of predictions with 95% confidence intervals for pairs with different authors is given in Figure 5.2. The plot provides a zoomed-in version of different artists groups from Figure 5.1, and offers, at least upon initial inspection, somewhat unintuitive results. Mean prediction for pairs of unseen artists is lower than that of seen artists, and again we suspect the unsupervised same class verification technique. Despite input images of unseen artists, the model is forced to predict one of the classes from the training set, resulting in a more diffuse class distribution vector, and in turn, a lower dot product prediction.

5.2 Vermeer–van Meegeren Comparison

An ingenious Dutch forger Han van Meegeren was present in the test set for better assessment of performance on more challenging examples. The forger has imitated some of the world's most famous artists' work, including the paintings of Johannes Vermeer. On the right half of Figure 5.3 is an example of a painting considered a genuine Vermeer, alongside an authentic painting for comparison.

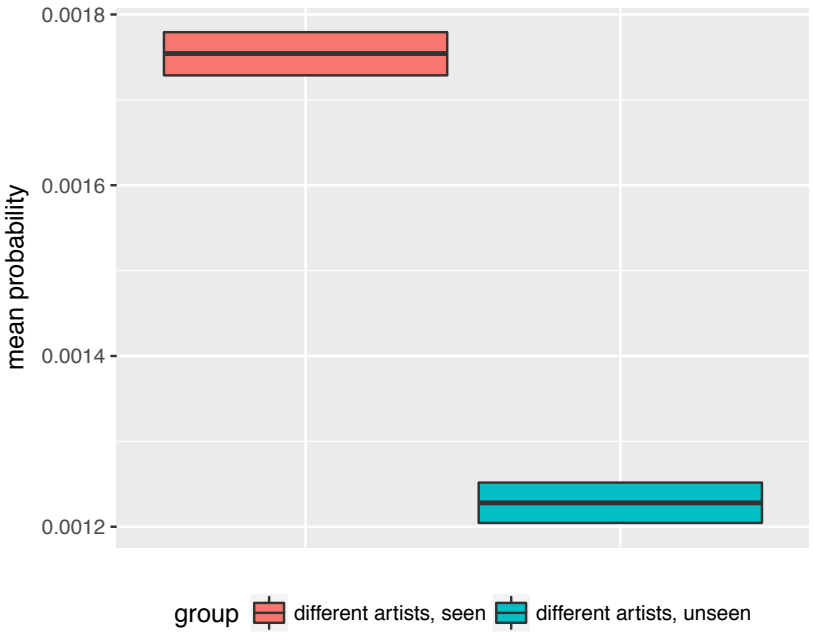


Figure 5.2: Posterior means with 95% confidence intervals of probability that two paintings are by the same artist. Pairs are split into groups based on whether painters were present in the training set (seen) or not present in the training set (unseen).

Figure 5.3: An example of similar-style paintings by Vermeer and van Meegeren. Johannes Vermeer: The Milkmaid (left) and Han van Meegeren: Supper at Emmaus (right). Supper at Emmaus was considered a genuine Vermeer by famous art experts like Abraham Bredius and M. Jean Decoen.[1]

Test set incorporates 20 paintings by Han van Meegeren and 13 paintings by Johannes Vermeer, generating 528 pairs for examination. Again, we offer visual comparison of posterior means with 95% confidence intervals of probability that both images are by the same artist, shown in Figure 5.4. Parameter estimation framework is identical to that in the previous section, where a normal sampling model without prior knowledge is used alongside an MCMC sampler [23]. Diagnostics are available in Appendix B. Pairs are split into groups based on whether one or both of the paintings were painted by Vermeer or van Meegeren. We are confident that mean prediction for the group where both paintings are by Vermeer is higher than for the group with different artists. On the other hand, probabilities of pairs with two Meegerens are closer to those with different artists. The forger indeed exploited styles of various painters, but the main reason for lower predictions is that works of Han van Meegeren were not present in the training set.

The plot in Figure 5.5 depicts fraction of predictions higher than a certain threshold, for all values of the threshold between the minimum and the maximum predicted probability. In particular, we are interested in pairs with the famous Supper at Emmaus from Figure 5.3 compared to authentic paintings by Johannes Vermeer. There are 13 aforestated pairs, one having a specifically high prediction, as suggested by the blue graph's plateau. Conflicting painting by Vermeer is The Geographer and is displayed on the left side of Figure 5.6. The probability that the two paintings are by the same painter for this pair is almost 13%, which is more than for any Vermeer - Vermeer pair in the test set.

5.3 Embedding

ConvNets transform the space of input images into a lower dimensional manifold in which examples are linearly separable, hence a linear classifier positioned at the top of the network. Trained models can straightforwardly be

[1]Images from `https://www.wikiart.org`.

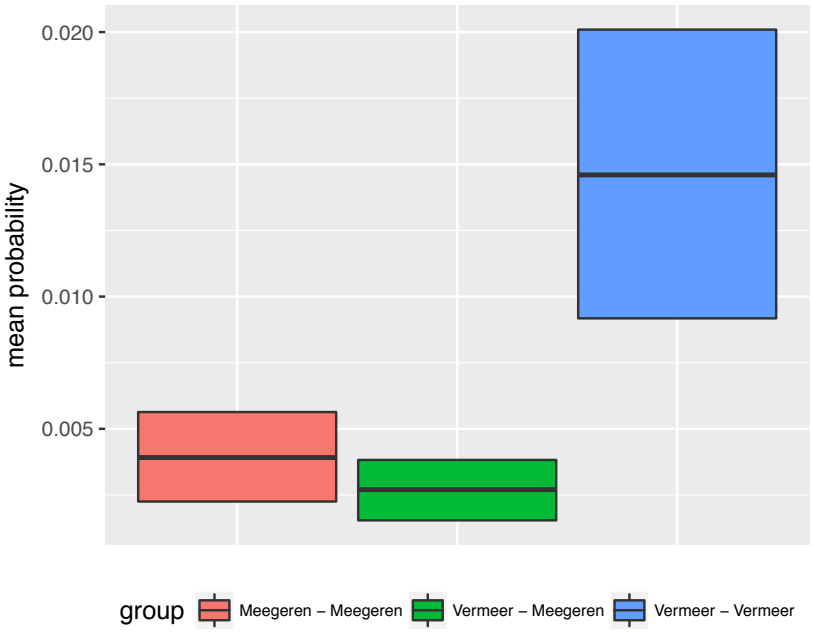


Figure 5.4: Posterior means with 95% confidence intervals of probability that two paintings are by the same artist. Pairs are split into groups based on whether one or both of the paintings were painted by Vermeer or van Meegeren.

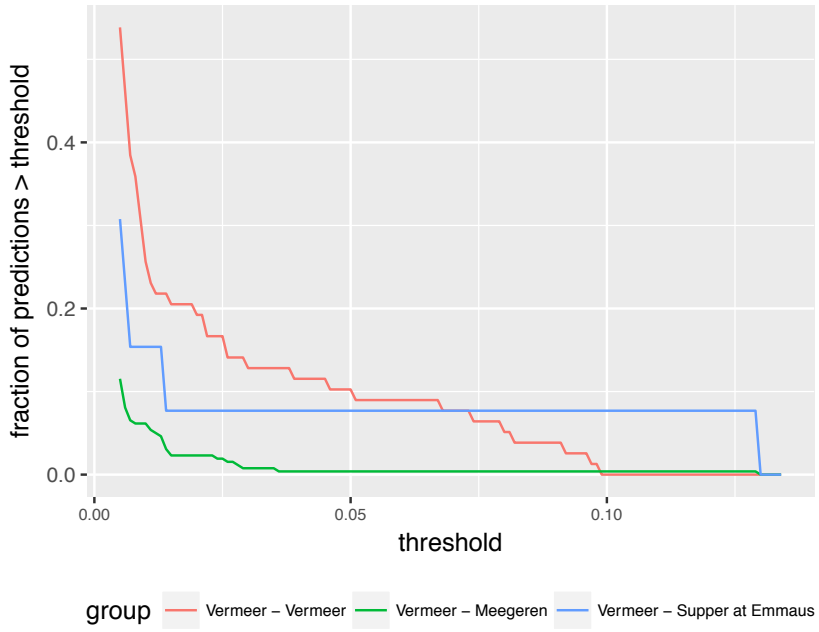


Figure 5.5: The fraction of predictions (probabilities) that are higher than a certain threshold value. Pairs are split into groups based on whether both of the paintings were painted by Vermeer or one by Vermeer and one by Meegeren. Additionally, a group of pairs with some Vermeer and Meegeren's Supper at Emmaus is shown.

Figure 5.6: Johannes Vermeer's The Geographer (left) compared to Han van Meegeren's Supper at Emmaus (right). Model's probability that the two paintings are by the same painter is 0.129748, which is more than for any Vermeer - Vermeer pair in the test set.[1]

converted into feature extractors by taking output vectors of the penultimate layer. Here we introduce a tool that simplifies the embedding process by deploying a trained *14 conv layers* network to a remotely accessible service, and expose it in Orange, an open source machine learning and data visualization software [5]. Example usage via a graphical user interface can be seen in Figure 5.7. The main advantage of integrating a pre-trained model into Orange is that it can be used in combination with existing data mining algorithms in the toolbox. We demonstrate the pragmatism by visualizing embedded features of Vermeers and van Meegerens, and by comparing the *14 conv layers* model to the *Inception-v3* model trained on a large ImageNet dataset [24].

Goodness of embedded features can qualitatively be evaluated by displaying Vermeer's and van Meegeren's paintings from the test set on a scatter plot seen in Figure 5.8. Principal Component Analysis is used to reduce dimensionality of 2048-dimensional vectors to two dimensions, which are then used for each image's coordinates. Four selected paintings on scatter plot are displayed in the Image Viewer widget on the right side of the figure.

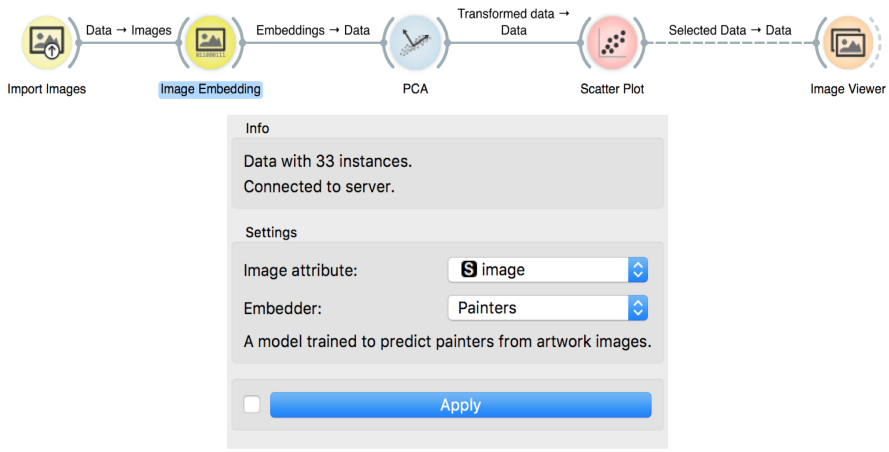


Figure 5.7: Example usage of the Image Embedding widget in Orange open source machine learning and data visualization software. The model that was trained to predict painters from artwork images is one of the available choices in the selection of the embedders.

Next we illustrate how embedded descriptors can be used in a supervised learning pipeline. We compare the *14 conv layers* model to the *Inception-v3* [24] model in a quantitative manner, by evaluating a Logistic Regression classifier, trained on both models' features, on 7,128 images from the test set. We perform a stratified 5-fold cross validation without hyperparameter tuning and give the results in Table 5.2. Our ConvNet clearly outperforms the *Inception-v3* model, confirming its suitability for tasks related to artwork images.

5.4 Identifying van Gogh's Paintings

G. Folego et al. [8] provides a public dataset of 333 high-resolution, density-normalized van Gogh and similar to van Gogh images, along with a train-test split used in their research. The training set consists of 264 images and test set is composed of 67 images. They divide each painting into smaller patches, extract features from each patch by employing a pre-trained ConvNet, and finally classify each area with a linear Support Vector Machine classifier.

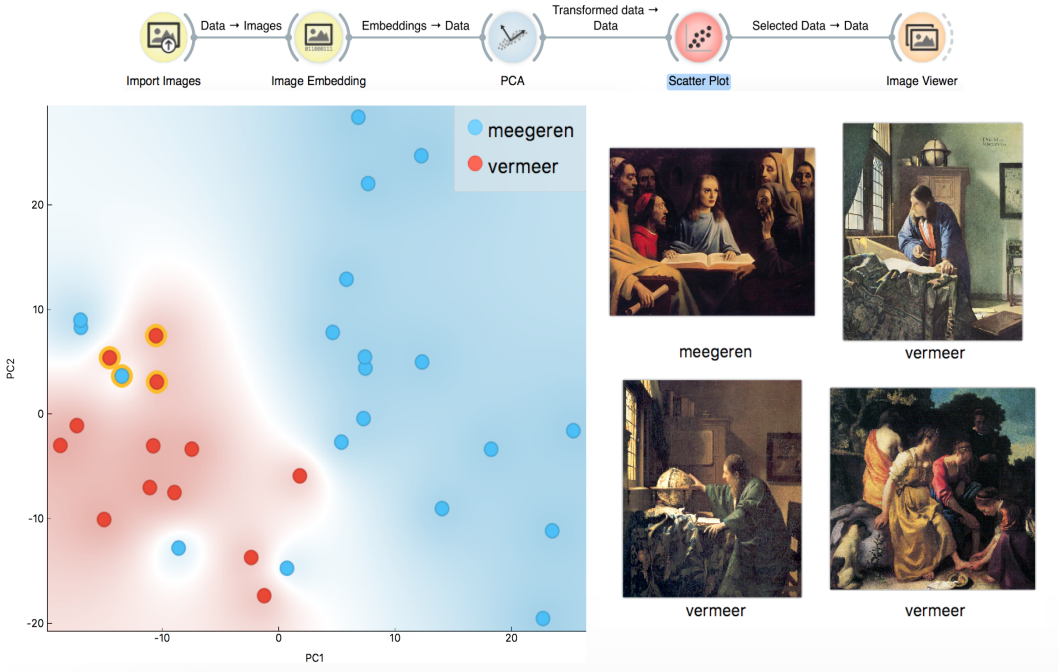


Figure 5.8: Scatter plot of Vermeer's and van Meegeren's paintings from the test set in Orange open source machine learning and data visualization software. Four selected paintings on scatter plot are displayed in the Image Viewer widget. First two principal components of 2048-dimensional embeddings are used for each image's coordinates.

model	test set AUC	test set CA
14 conv layers	0.928	0.351
Inception-v3	0.878	0.215

Table 5.2: AUC and classification accuracy of a Logistic Regression classifier, trained on features from ConvNets, for a multi-class painter recognition task. 5-fold cross validation was performed on 7,128 artwork images from the test set.

model	AUC	CA	F1
14 conv layers	0.972	0.896	0.851
Inception-v3	0.909	0.881	0.840
G. Folego et al.	-	-	0.880, 0.906, 0.923

Table 5.3: AUC and classification accuracy of a Logistic Regression classifier, trained on features from ConvNets, for identifying van Gogh's paintings. The F1 score is given for comparison to G. Folego et al.

They further propose multiple aggregation methods for combining all patch scores into a final prediction, indicating whether painting was created by van Gogh or not.

We trained a Logistic Regression Classifier on embeddings of training images using the same split as G. Folego et al., and tuned hyperparameters using 3-fold cross validation. Comparison of performance on the test set is given in Table 5.3. Our method's F1 score is only moderately higher than *Inception-v3*'s and 0.029 lower than worst-performing aggregation method, suggested by G. Folego et al. Interestingly, features from *14 conv layers* model outperform *Inception-v3*'s and best-performing method by G. Folego et al. in terms of False Positive Rate (FPR), despite negative samples being amid the most difficult, since they are closely related to van Gogh [8]. Confusion matrices confirming our method's advantage at type I errors are depicted in Table 5.4, where van Gogh paintings are labeled as *vg* and non-van Goghs as *nvg*.

We hypothesize that our F1 score is lower due to a modicum of information that our approach exploits. For example, G. Folego et al. trained their classifiers on 47,408 224×224 pixel patches, whereas we only use a single 256×256 patch for each of the 264 images in the training set. On the other hand, False Positive Rate comparison provides crucial evidence that constructing discriminative features from low-resolution images is realistic and should not be disregarded.

14 conv layers

actual class	vg	nvg
vg	20	5
nvg	2	40

Inception-v3

	vg	nvg
vg	21	4
nvg	4	38

G. Folego et al.

	vg	nvg
vg	24	1
nvg	3	39

Table 5.4: Confusion matrices of a Logistic Regression classifier, trained on each model's outputs of the penultimate layer, for recognizing van Gogh's paintings. In addition to that, a confusion matrix of the best-performing method by G. Folego et al. is given for comparison. Van Goghs ara labeled as *vg* and non-van Goghs as *nvg*.

Chapter 6

Conclusion

Painting authentication of digital artwork images is an open research problem. In this work, we developed a general technique capable of identifying a large number of artists from digitized paintings and demonstrated its applicability in related tasks. Ignoring low-level characteristics enabled us to exploit a much larger dataset composed of low-resolution images. We conceptualize our proposed algorithm to be used alongside techniques established in related work, and thus further augment the art of connoisseurship.

We evaluated our models in a Kaggle competition by classifying pairs of images based on their authors. Our approach obtained 0.92890 AUC, a top score among the competitors, and distinguished itself in differentiation between Vermeer's and van Meegeren's paintings. We further experimented with related tasks by employing transfer learning and showed that themes and motifs available in low-resolution images hold important discriminative capabilities.

Despite encouraging results, there is still room for improvement. We believe that training and end-to-end class verification model would reflect superior performance at the task of classifying pairs of images. Additionally, we believe that using state-of-the-art convolutional neural network architectures, in combination with fine-tuning models pre-trained on large datasets, would also enhance our proposed approach. In addition to that, our work

poses an important question: how to combine low-level information with semantic features, when high-resolution scans of paintings are accessible to a practitioner.

Appendix A

Model Architectures

This appendix contains architectures and Keras[1] code samples of individual convolutional neural networks trained during a hyperparameter search. Comparison of results for the multi-class painter recognition task is available in Table 4.1. Architecture of the best performing, fourteen convolutional layers model is not given in this appendix, but is available in Chapter 4. In Listing A.1 auxiliary functions for building ConvNet architectures are given.

```
def _convolutional_layer(nb_filter, input_shape=None):
    if input_shape:
        return _first_convolutional_layer(nb_filter, input_shape)
    else:
        return _intermediate_convolutional_layer(nb_filter)


def _first_convolutional_layer(nb_filter, input_shape):
    return Conv2D(
        nb_filter=nb_filter, nb_row=3, nb_col=3, input_shape=input_shape,
        border_mode='same', init='he_normal', W_regularizer=l2(l=0.003))


def _intermediate_convolutional_layer(nb_filter):
    return Conv2D(
```

[1]Deep Learning library available on `https://keras.io`.

```
            nb_filter=nb_filter, nb_row=3, nb_col=3, border_mode='same',
            init='he_normal', W_regularizer=l2(l=0.003))


    def _dense_layer(output_dim):
        return Dense(
            output_dim=output_dim, W_regularizer=l2(l=0.003),
            init='he_normal')
```

Listing A.1: Auxiliary functions for building ConvNet architectures. All convolutional layers have 3×3 filters with stride 1.

```
model = Sequential()
model.add(_convolutional_layer(nb_filter=16, input_shape=imgs_dim))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=16))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
```

```
model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=128))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=128))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=128))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=256))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=256))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=256))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=512))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=512))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=512))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(p=0.5))
model.add(Flatten())
model.add(_dense_layer(output_dim=2048))
model.add(BatchNormalization())
model.add(PReLU(init='he_normal'))
model.add(Dropout(p=0.5))
model.add(_dense_layer(output_dim=1584))
model.add(BatchNormalization())
model.add(Activation(activation='softmax'))
```

Listing A.2: Code for a seventeen convolutional layers model.

```
model = Sequential()
model.add(_convolutional_layer(nb_filter=16, input_shape=imgs_dim))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=16))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
```

```
model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=128))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=128))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=128))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=256))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=256))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=256))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(p=0.5))
model.add(Flatten())
model.add(_dense_layer(output_dim=2048))
model.add(BatchNormalization())
model.add(PReLU(init='he_normal'))
model.add(Dropout(p=0.5))
model.add(_dense_layer(output_dim=1584))
model.add(BatchNormalization())
model.add(Activation(activation='softmax'))
```

Listing A.3: Code for a fourteen convolutional layers model.

```
model = Sequential()
model.add(_convolutional_layer(nb_filter=16, input_shape=imgs_dim))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=16))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=128))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=128))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=128))
```

```
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(p=0.5))
model.add(Flatten())
model.add(_dense_layer(output_dim=2048))
model.add(BatchNormalization())
model.add(PReLU(init='he_normal'))
model.add(Dropout(p=0.5))
model.add(_dense_layer(output_dim=1584))
model.add(BatchNormalization())
model.add(Activation(activation='softmax'))
```

Listing A.4: Code for an eleven convolutional layers model.

```
model = Sequential()
model.add(_convolutional_layer(nb_filter=16, input_shape=imgs_dim))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=16))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=32))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
```

```
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(_convolutional_layer(nb_filter=64))
model.add(BatchNormalization(axis=1))
model.add(PReLU(init='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(p=0.5))
model.add(Flatten())
model.add(_dense_layer(output_dim=2048))
model.add(BatchNormalization())
model.add(PReLU(init='he_normal'))
model.add(Dropout(p=0.5))
model.add(_dense_layer(output_dim=1584))
model.add(BatchNormalization())
model.add(Activation(activation='softmax'))
```

Listing A.5: Code for an eight convolutional layers model.

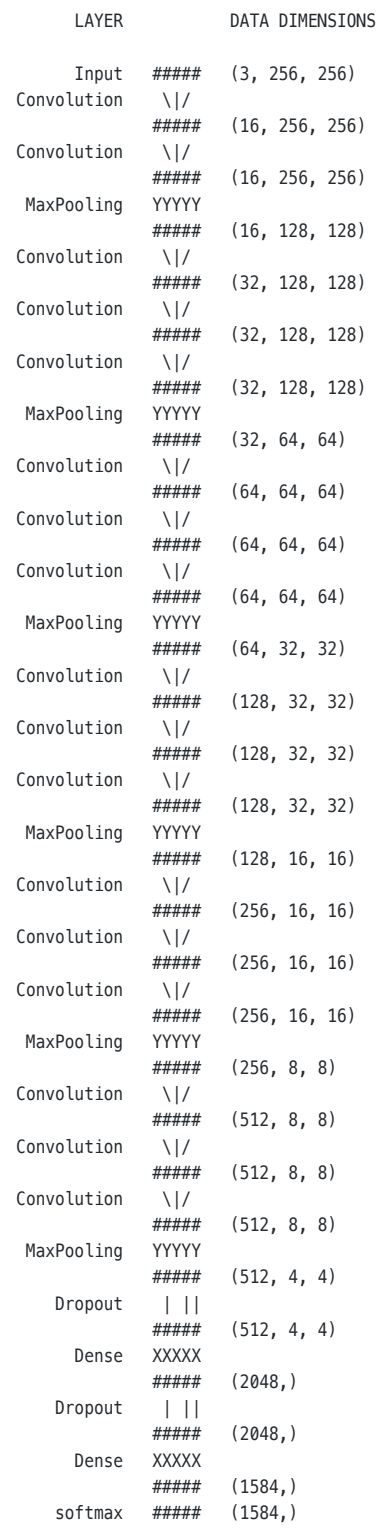


Figure A.1: Seventeen convolutional layers model architecture with non-linearities and batch normalization layers omitted.

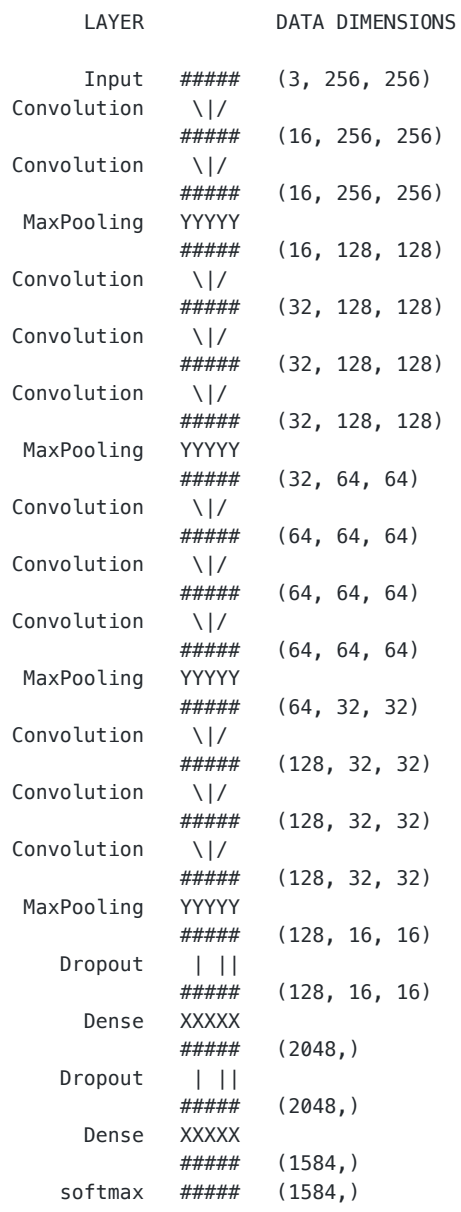


Figure A.2: Eleven convolutional layers model architecture with non-linearities and batch normalization layers omitted.

```
      LAYER          DATA DIMENSIONS

      Input   #####   (3, 256, 256)
Convolution    \|/
              #####   (16, 256, 256)
Convolution    \|/
              #####   (16, 256, 256)
 MaxPooling   YYYYY
              #####   (16, 128, 128)
Convolution    \|/
              #####   (32, 128, 128)
Convolution    \|/
              #####   (32, 128, 128)
 MaxPooling   YYYYY
              #####   (32, 64, 64)
Convolution    \|/
              #####   (64, 64, 64)
Convolution    \|/
              #####   (64, 64, 64)
 MaxPooling   YYYYY
              #####   (64, 32, 32)
Convolution    \|/
              #####   (128, 32, 32)
Convolution    \|/
              #####   (128, 32, 32)
 MaxPooling   YYYYY
              #####   (128, 16, 16)
    Dropout    | ||
              #####   (128, 16, 16)
      Dense   XXXXX
              #####   (2048,)
    Dropout    | ||
              #####   (2048,)
      Dense   XXXXX
              #####   (1584,)
    softmax   #####   (1584,)
```

Figure A.3: Eight convolutional layers model architecture with non-linearities and batch normalization layers omitted.

Appendix B

MCMC Diagnostics

This appendix contains traceplots and summaries for diagnostic purposes of MCMC sampling used in Chapter 5. Each figure caption includes a Monte Carlo standard error, effective sample size due to autocorrelation and the R-hat convergence statistic. It seems—based on traceplots and statistics—that all chains have converged.

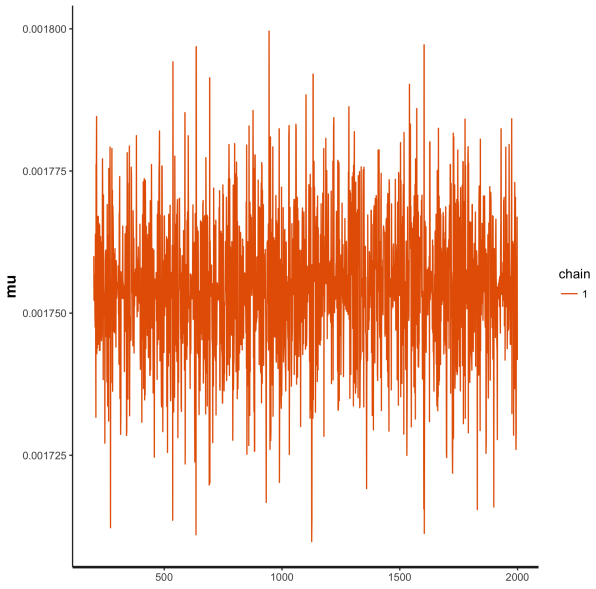


Figure B.1: Traceplot of mean prediction sampling (different artists, seen). Standard error: 3.02e-07, effective sample size: 1790.93, $\widehat{R}$: 1.002.

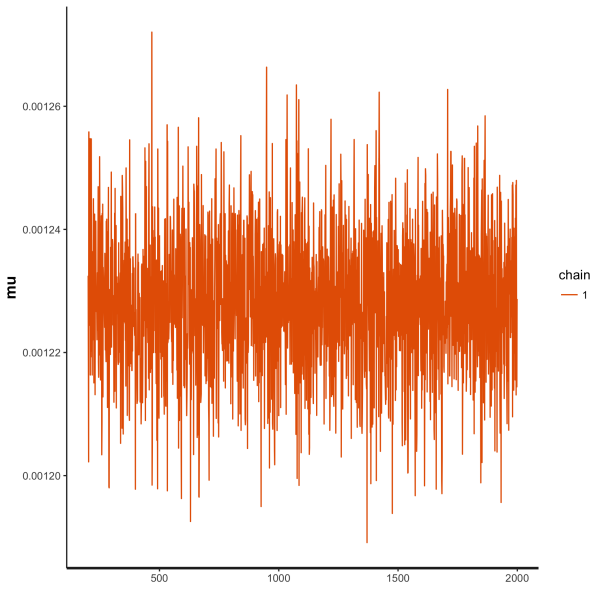


Figure B.2: Traceplot of mean prediction sampling (different artists, unseen). Standard error: 2.82e-07, effective sample size: 1800.00, $\widehat{R}$: 0.999.

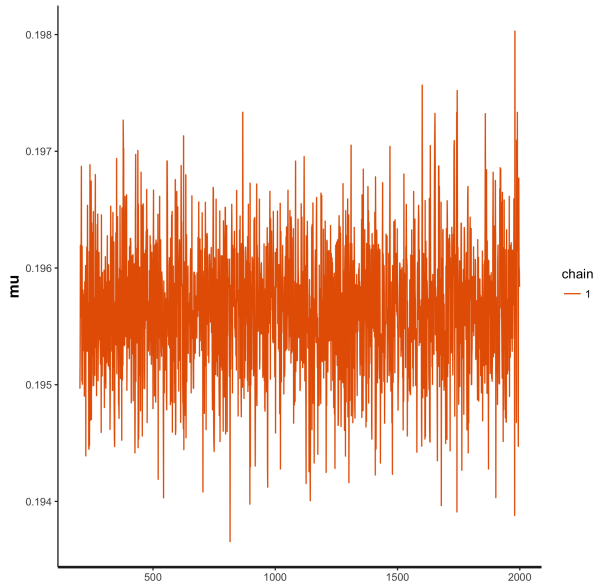


Figure B.3: Traceplot of mean prediction sampling (same artist, seen). Standard error: 1.41e-05, effective sample size: 1672.04, $\widehat{R}$: 0.999.

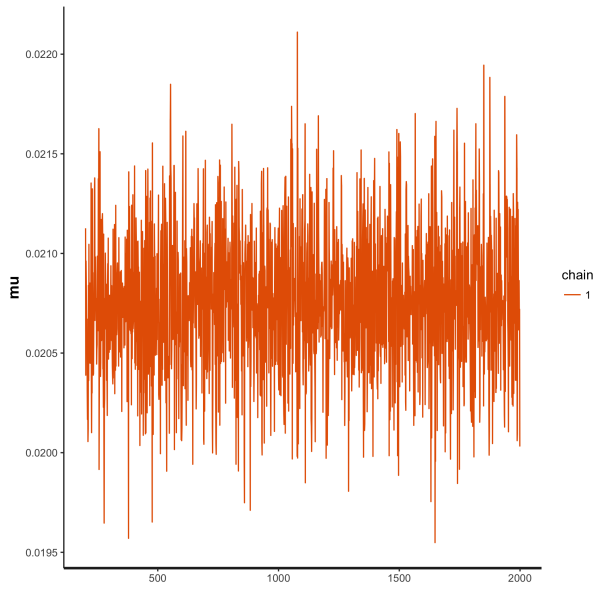


Figure B.4: Traceplot of mean prediction sampling (same artist, unseen). Standard error: 8.69e-06, effective sample size: 1698.56, $\widehat{R}$: 0.999.

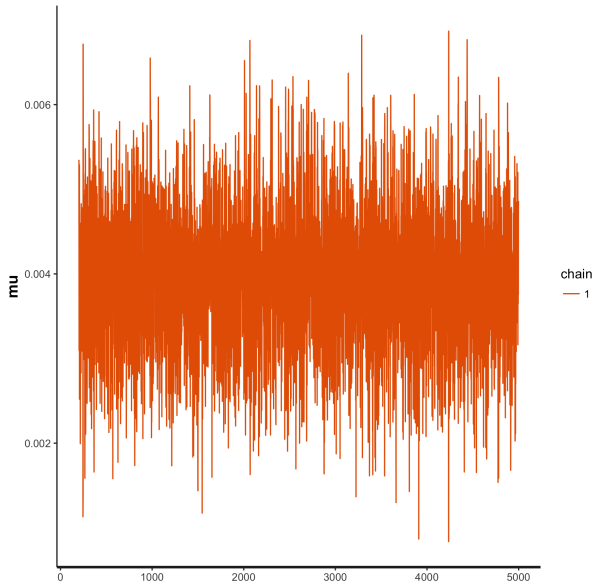


Figure B.5: Traceplot of mean prediction sampling (Meegeren - Meegeren pairs). Standard error: 1.30e-05, effective sample size: 4280.65, $\widehat{R}$: 0.999.

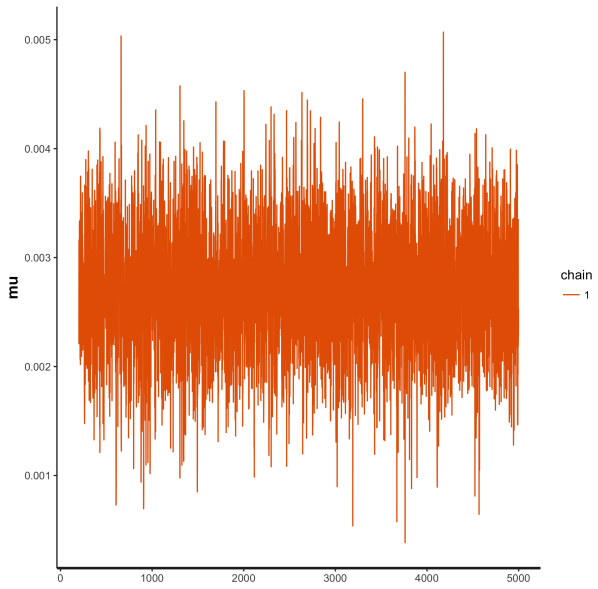


Figure B.6: Traceplot of mean prediction sampling (Vermeer - Meegeren pairs). Standard error: 8.61e-06, effective sample size: 4800.00, $\widehat{R}$: 1.000.

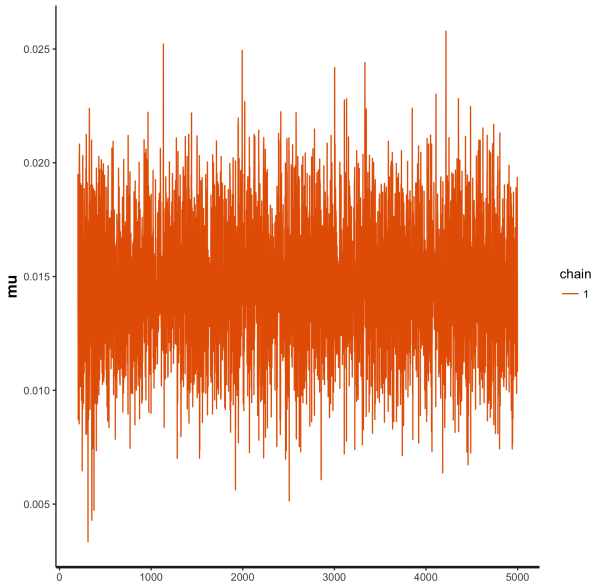


Figure B.7: Traceplot of mean prediction sampling (Vermeer - Vermeer pairs). Standard error: 4.22e-05, effective sample size: 4415.35, $\widehat{R}$: 0.999.

Bibliography

[1] D. Throsby, The production and consumption of the arts: A view of cultural economics, Journal of Economic Literature 32 (1) (1994) 1–29.

[2] G. Newman, P. Bloom, Art and authenticity: The importance of originals in judgments of value, Journal of Experimental Psychology 141 (2011) 558–69.

[3] B. Cornelis, A. Dooms, I. Daubechies, P. Schelkens, Report on digital image processing for art historians, in: SAMPTA, 2009, pp. Special–session.

[4] C. R. Johnson, E. Hendriks, I. J. Berezhnoy, E. Brevdo, S. M. Hughes, I. Daubechies, J. Li, E. Postma, J. Z. Wang, Image processing for artist identification, IEEE Signal Processing Magazine 25 (4) (2008) 37–48.

[5] J. Demšar, T. Curk, A. Erjavec, Črt Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, B. Zupan, Orange: Data mining toolbox in python, Journal of Machine Learning Research 14 (2013) 2349–2353.

[6] S. Karayev, A. Hertzmann, H. Winnemoeller, A. Agarwala, T. Darrell, Recognizing image style, CoRR abs/1311.3715.
URL `http://arxiv.org/abs/1311.3715`

[7] J. Li, L. Yao, E. Hendriks, J. Z. Wang, Rhythmic brushstrokes distinguish van Gogh from his contemporaries: findings via automated brush-

stroke extraction, IEEE Transactions on Pattern Analysis and Machine Intelligence 34 (6) (2012) 1159–1176.

[8] G. Folego, O. Gomes, A. Rocha, From impressionism to expressionism: Automatically identifying van Gogh's paintings, in: Image Processing (ICIP), 2016 IEEE International Conference on, IEEE, 2016, pp. 141–145.

[9] S. Lyu, D. Rockmore, H. Farid, A digital technique for art authentication, Proceedings of the National Academy of Sciences of the United States of America 101 (49) (2004) 17006–17010.

[10] L. Shamir, T. Macura, N. Orlov, D. M. Eckley, I. G. Goldberg, Impressionism, expressionism, surrealism: Automated recognition of painters and schools of art, ACM Transactions on Applied Perception (TAP) 7 (2) (2010) 8.

[11] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[12] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[13] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 315–323.

[14] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[15] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of

the IEEE international conference on computer vision, 2015, pp. 1026–1034.

[16] C. M. Bishop, Pattern recognition and machine learning, Springer, 2006.

[17] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, CoRR abs/1409.1556.
URL `http://arxiv.org/abs/1409.1556`

[18] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, L. D. Jackel, Handwritten digit recognition with a back-propagation network, in: Advances in neural information processing systems, 1990, pp. 396–404.

[19] A. Krogh, J. A. Hertz, A simple weight decay can improve generalization, in: Advances in neural information processing systems, 1992, pp. 950–957.

[20] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting., Journal of Machine Learning Research 15 (1) (2014) 1929–1958.

[21] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, 2015, pp. 448–456.

[22] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[23] B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, A. Riddell, Stan: A probabilistic programming language, Journal of Statistical Software, Articles 76 (1) (2017) 1–32.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the

IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.

[25] D. G. Stork, Computer Vision and Computer Graphics Analysis of Paintings and Drawings: An Introduction to the Literature, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 9–24.

[26] M. Oquab, L. Bottou, I. Laptev, J. Sivic, Learning and transferring mid-level image representations using convolutional neural networks, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1717–1724.

[27] A. S. Razavian, H. Azizpour, J. Sullivan, S. Carlsson, CNN features off-the-shelf: an astounding baseline for recognition, CoRR abs/1403.6382. URL `http://arxiv.org/abs/1403.6382`

[28] J. Zujovic, L. Gandy, S. Friedman, B. Pardo, T. N. Pappas, Classifying paintings by artistic genre: An analysis of features & classifiers, in: Multimedia Signal Processing, 2009. MMSP. IEEE International Workshop on, IEEE, 2009, pp. 1–5.

[29] R. S. Arora, A. Elgammal, Towards automated classification of fine-art painting style: A comparative study, in: Pattern Recognition (ICPR), 21st International Conference on, IEEE, 2012, pp. 3541–3544.

[30] Y. Bar, N. Levy, L. Wolf, Classification of artistic styles using binarized features derived from a deep neural network., in: ECCV Workshops, 2014, pp. 71–84.

[31] I. Berezhnoy, E. Postma, J. van den Herik, Computer analysis of van Gogh's complementary colours, Pattern Recognition Letters 28 (6) (2007) 703–709.

[32] E. Cetinic, S. Grgic, Automated painter recognition based on image feature extraction, in: ELMAR, 2013 55th International Symposium, IEEE, 2013, pp. 19–22.

[33] T. Mensink, J. Van Gemert, The rijksmuseum challenge: Museum-centered visual recognition, in: Proceedings of International Conference on Multimedia Retrieval, ACM, 2014, p. 451.

[34] F. S. Khan, S. Beigpour, J. Van de Weijer, M. Felsberg, Painting-91: a large scale database for computational painting categorization, Machine vision and applications 25 (6) (2014) 1385–1397.

[35] L. A. Gatys, A. S. Ecker, M. Bethge, A neural algorithm of artistic style, arXiv preprint arXiv:1508.06576.

[36] J. Johnson, A. Alahi, L. Fei-Fei, Perceptual losses for real-time style transfer and super-resolution, in: European Conference on Computer Vision, Springer, 2016, pp. 694–711.

[37] V. Dumoulin, J. Shlens, M. Kudlur, A. Behboodi, F. Lemic, A. Wolisz, M. Molinaro, C. Hirche, M. Hayashi, E. Bagan, et al., A learned representation for artistic style, arXiv preprint arXiv:1610.07629.

[38] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, E. Shechtman, Controlling perceptual factors in neural style transfer, arXiv preprint arXiv:1611.07865.

[39] D. Ulyanov, V. Lebedev, A. Vedaldi, V. S. Lempitsky, Texture networks: Feed-forward synthesis of textures and stylized images., in: ICML, 2016, pp. 1349–1357.

[40] A. L. Maas, A. Y. Hannun, A. Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: Proc. ICML, Vol. 30, 2013.

[41] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, Y. LeCun, The loss surfaces of multilayer networks, in: Artificial Intelligence and Statistics, 2015, pp. 192–204.

[42] C. Andrieu, N. De Freitas, A. Doucet, M. I. Jordan, An introduction to mcmc for machine learning, Machine Learning 50 (1-2) (2003) 5–43.

[43] S. Chopra, R. Hadsell, Y. LeCun, Learning a similarity metric discriminatively, with application to face verification, in: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, Vol. 1, IEEE, 2005, pp. 539–546.

[44] Y. Taigman, M. Yang, M. Ranzato, L. Wolf, Deepface: Closing the gap to human-level performance in face verification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1701–1708.